The Girard-Reynolds Isomorphism

Philip Wadler

Avaya Labs, wadler@avaya.com

Abstract. The second-order polymorphic lambda calculus, F2, was independently discovered by Girard and Reynolds. Girard additionally proved a *representation theorem*: every function on natural numbers that can be proved total in second-order intuitionistic propositional logic, P2, can be represented in F2. Reynolds additionally proved an *abstraction theorem*: for a suitable notion of logical relation, every term in F2 takes related arguments into related results. We observe that the essence of Girard's result is a projection from P2 into F2, and that the essence of Reynolds's result is an embedding of F2 into P2, and that the Reynolds embedding followed by the Girard projection is the identity. The Girard projection discards all first-order quantifiers, so it seems unreasonable to expect that the Girard projection followed by the Reynolds embedding should also be the identity. However, we show that in the presence of Reynolds's *parametricity* property that this is indeed the case, for propositions corresponding to inductive definitions of naturals, products, sums, and fixpoint types.

1 Introduction

Double-barreled names in science may be special for two reasons: some belong to ideas so subtle that they required two collaborators to develop; and some belong to ideas so sublime that they possess two independent discoverers. The Curry-Howard isomorphism is an idea of the first sort that guarantees the existence of ideas of the second sort, such as the Hindley-Milner type system and the Girard-Reynolds polymorphic lambda calculus.

The Curry-Howard isomorphism consists of a correspondence between a logical calculus and a computational calculus. Each logical formula corresponds to a computational type, each logical proof corresponds to a computational term, and reduction of proofs corresponds to reductions of terms. This last point means that it is not just formulas and proofs that are preserved by the correspondence, but the structure between them as well; hence we have no mere bijection but a true isomorphism.

Curry formulated this principle for combinatory logic and combinator terms [CF58]. Howard observed that it also applies to intuitionistic propositional logic and simply-typed lambda terms [How80]. The same idea extends to a correspondence between first-order intuitionistic logic with propositional variables and simply-typed lambda calculus with type variables, which explains why the logician Hindley and the computer scientist Milner independently discovered the Hindley-Milner type system [Hin69,Mil78,DM82]. It also extends to a

correspondence between second-order intuitionistic logic with quantifiers over proposition variables and second-order typed lambda calculus with quantifiers over type variables, which explains why the logician Girard and the computer scientist Reynolds independently discovered the polymorphic lambda calculus [Gir72,Rey74].

Girard and Reynolds each made additional discoveries about the calculus that bears their name, henceforth referred to as F2. Girard proved a *representation theorem*: every function on natural numbers that can be proved total in second-order predicate calculus P2 (with both first- and second-order quantifiers) can be represented in F2 (using second-order quantifiers only). Reynolds proved an *abstraction theorem*: for a suitable notion of logical relation, every term in F2 takes related arguments into related results [Rey83].

The calculus P2 is larger than the image under the Curry-Howard isomorphism of F2: the former has first-order terms (we will take these to be terms of untyped lambda calculus) and both first- and second-order quantifiers, while the latter has second-order quantifiers only. Nonetheless, the essence of Girard's result is a projection from P2 onto F2 that is similar to the Curry-Howard isomorphism, in that it takes formulas to types and proofs to terms, but differs in that it erases all information about first-order terms and first-order quantifiers. This mapping also preserves reductions, so it is no mere surjection but a true homomorphism.

Reynolds's result traditionally concerns binary relations, but it extends to other notions of relation, including a degenerate unary case. In the unary version, the essence of Reynolds's result is an embedding from F2 into P2 that is similar to the Curry-Howard isomorphism, in that it takes types to formulas and proofs to terms, but differs in that it adds information about first-order quantifiers and first-order terms. This mapping also preserves reductions, so it is no mere injection but a true homomorphism. Furthermore, the result on binary relations can be recovered from the result on unary relations by a doubling operation, an embedding from P2 into P2 that takes formulas into formulas, proofs into proofs, and preserves reductions.

Strachey distinguished two types of polymorphism, where the meaning of a term depends upon a type [Str67]. In *parametric* polymorphism, the meaning of the term varies uniformly with the type (an example is the length function), while in *ad hoc* polymorphism, the meaning of the term at different types may not be related (an example is plus, which may have quite different meanings on integers, floats, and strings). Reynolds introduced a *parametricity* condition to capture a semantic notion corresponding to Strachey's parametric polymorphism. One consequence of the parametricity condition is the Identity Extension Lemma, which asserts that the relation corresponding to a type is the identity relation, so long as the relation corresponding to any free type variable is also taken to be the identity relation.

The Reynolds embedding followed by the Girard projection is the identity. Remarkably, I can find no place in the literature where this is remarked! While

reading between the lines suggests that some researchers have intuitively grasped this duality, its precise description seems to have been more elusive.

Going the other way, it seems unreasonable to expect that the Girard projection followed by the Reynolds embedding should also yield the identity, because the projection discards all information about first-order terms. For instance, here is the standard inductive definition of natural numbers in P2.

$$\mathsf{Nat} \;\equiv\; \{\, n \,|\, \forall Z.\, (\forall m.\, m \in Z \to \mathsf{succ}\; m \in Z) \to \mathsf{zero} \in Z \to n \in Z \,\}$$

Here succ and zero are the usual successor and zero operations on Church numerals in untyped lambda calculus. Applying the Girard projection yields $\mathbf{Nat} \;\equiv\; \mathsf{Nat}^\circ$, the type of the Church numerals in F2.

$$\mathbf{Nat} \;\equiv\; \forall Z.\, (Z \to Z) \to (Z \to Z)$$

Then applying the Reynolds embedding in turn yields the following predicate, back in P2.

$$\mathbf{Nat}^* \;\equiv\; \{\, n \,|\, \forall Z.\, \forall s.\, (\forall m.\, m \in Z \to s\; m \in Z) \to \forall z.\, z \in Z \to n\; s\; z \in Z \,\}$$

This predicate does not look much like Nat — it makes no mention of succ or zero. However we will see that if we assume that the type **Nat** satisfies an analogue of Reynolds's *parametricity* condition, then Nat and $\mathbf{Nat}^*$ are equivalent, in that one can prove that any term satisfying the first predicate also satisfies the second, and conversely.

Hence, in the presence of parametricity, not only does the Girard projection take Nat to **Nat**, but also the Reynolds embedding takes **Nat** to Nat, and so in this important case one has not merely an embedding-projection pair but a true isomorphism.

We will show that similar results hold for all algebraic types, those types built from products, sums, and fixpoints. An alternative characterization of algebraic types is given by Böhm and Berarducci, where they are shown to be equivalent to the types of rank two with all quantifiers on the outside [BB85].

This paper contains a version of Reynold's Abstraction Theorem, but only hints at Girard's Representation Theorem. For a complete exposition of the latter, consult the excellent tutorials by Girard, Taylor, and Lafont [GLT89] or by Leivant [Lei90].

Both Girard's and Reynold's results have spawned large bodies of related work. The representation of algebraic data types in polymorphic lambda calculus was proposed by Böhm and Berarducci [BB85]. Girard's representation theorem is has been further explored by Leivant [Lei83,Lei90] and by Krivine and Parigot [KP90], among others. Reynolds's parametricity has been further explored by Reynolds [Rey84,Rey90,MR91], Reynolds and Plotkin [RP90], Bainbridge, Freyd, Scedrov and Scott [BFSS90], Hasegawa [Has94], Pitts [Pit87,Pit89,Pit98], and Wadler [Wad89,Wad91], among others. Formulations of the abstraction theorem in terms of logics have been examined by Mairson [Mai91], in various combinations by Abadi, Cardelli,

Curien, and Plotkin [ACC93,PA93,PAC94], and by Takeuti [Tak98]. Many of these works observe some connection between parametricity and algebraic types [RP90,Has94,BFSS90,ACC93,PA93,PAC94,Tak98]. Breazu-Tannen and Coquand [BC88], building on work of Moggi [Mog86], show how to turn any model of untyped lambda calculus into a model of polymorphic lambda calculus that satisfies a parametricity condition at all algebraic types.

The results here are not so much new proofs as old proofs clarified. In particular, we set Girard's and Reynolds's proofs in a common framework, highlighting the relationship between them. Unlike some previous work, no complex semantic formalism or specialized logic is required; all is formulated within the well-known system of second-order predicate logic. And teasing apart the relation between parametric and extensive types appears to be new.

Particularly strong influences on this work have been: the original work of Girard and Reynolds; Leivant [Lei90], who presents Girard's result as a projection from P2 to F2; Mairson [Mai91], who presents Reynolds's result as an embedding of F2 in P2; Plotkin and Abadi [PA93] and Takeuti [Tak98], who present typed analogues of the untyped logic used here; Krivine and Parigot [KP90], who use a logic over untyped lambda terms similar to P2.

Treating Girard's representation theorem requires use of a logic with untyped terms, since the whole point of the theorem is to demonstrate that functions defined without reference to types may be represented in a typed calculus. However, use of an untyped calculus does severely restrict the available models. A typed calculus, such as that considered by Plotkin and Abadi [PA93] or Takeuti [Tak98], allows a fuller range of models.

Mairson [Mai91] appears to have grasped the inverse relation between the Reynolds embedding and the Girard projection, though he does not quite manage to state it. However, Mairson does seem to have missed the power of parametricity. He mislabels as "parametricity" the analogue of Reynolds's Abstraction Theorem, and he never states an analogue of Reynolds's parametricity condition or the Identity Extension Lemma. Thus when he writes "proofs of these equivalences still seem to require structural induction, as well as stronger assumptions than parametricity" [Mai91], I believe this is misleading: the equivalences he refers to cannot be proved using the Abstraction Theorem alone, but can indeed be proved in the presence of parametricity.

The Curry-Howard isomorphism has informed the development of powerful lambda calculi with dependent types, such as de Bruijn's Automath [deB70], Martin-Löf's type theory [Mar82], Constable's Nuprl [Con86], Coquand and Huet's calculus of constructions [CH88], and Barendregt's lambda cube [Bar91]. Each of these calculi introduces dependent types (types that depend upon values) to map first-order quantifiers into the type system. In contrast, the Girard projection discards all first-order information. To quote Leivant [Lei90],

> we pursue a dual approach: rather than enriching the type systems to match logic, we impoverish logic to match the type structure.

What is remarkable is that even after this impoverishment enough power remains to capture much of what matters in computing: the algebraic types, such as naturals, products, sums, and fixpoints.

The remainder of this paper is organized as follows. Section 2 introduces the second-order lambda calculus F2 and the second-order logic P2. Section 3 describes the Reynolds embedding and the Girard projection, and observes that the embedding followed by the projection is the identity. Section 4 explains doubling and parametricity. Section 5 shows that the two definitions of the naturals are equivalent under the parametricity postulate, and similarly for other algebraic types, and hence that there is a sense in which the Girard projection followed by the Reynolds embedding is also the identity.

2 Second-order lambda calculus and logic

The second-order lambda calculus F2 is summarized in Figure 1, and second-order intuitionistic logic P2 is summarized in Figure 2. For each calculus we list the syntactic categories, the proof rules, and the reductions that act upon proofs. We deliberately use x, y, z to range over individual variables in both calculi, and use X, Y, Z to range over type variables in F2 and predicate variables in P2. We write $\equiv$ for syntactic equivalence of terms, formulas, or proofs.

Judgements in F2 have the form $\Gamma \vdash t : A$, expressing that term t has type A in context Γ, where a context consists of pairs $x : A$ associating individual variables with types. Types are formed from type variables X, functions $A \to B$, and quantification over types $\forall X.\, B$. Terms are formed from individual variables x, abstraction $\lambda x : A.\, u$, application $s\, t$, type abstraction $\Lambda X.\, u$, and type application $s\, A$. A proof δ uniquely determines its concluding judgement $\Gamma \vdash t : A$, and conversely, δ is uniquely determined by Γ and t. A proof reduces when an introducer is followed by the corresponding eliminator. We write $\delta[x := \epsilon]$ for the proof that results by substituting proof ϵ for each use of (Id) on x in the proof δ.

Judgements in P2 have the form $\Theta \vdash \phi$, expressing that the formula list Θ has the formula ϕ as a consequence. An atomic formula has the form $M \in X$, where M is a term and X is a predicate variable. Fomulas are formed from atomic formulas, implication $\phi \to \psi$, quantification over individual variables $\forall x.\, \psi$, and quantification over predicate variables $\forall X.\, \psi$.

A term is an untyped lambda term. We write $M =_\beta N$ if M and N can be shown equivalent by β reduction.

Predicate variables X range over properties of terms. Notationally, we treat these as sets. Thus we write $M \in X$ to mean that term M satisfies predicate X. The comprehension notation $\{\, x \mid \phi \,\}$ denotes the predicate of x that is satisfied when the formula ϕ over x holds; so x is free in ϕ but bound in $\{\, x \mid \phi \,\}$. We write $\psi[X := \{\, x \mid \phi \,\}]$ for the formula that results by replacing each occurrence of an atomic formula $M \in X$ in ψ by the formula $\phi[x := M]$. (Some formulations write $X(M)$ or $X\, M$ instead of $M \in X$, and $(x)\phi$ or $x.\phi$ instead of $\{\, x \mid \phi \,\}$.)

It will prove convenient to have a notation for arbitrary predicates. We let P and Q range over comprehensions of the form $\{\, x \mid \phi \,\}$, and write $M \in P$ as shorthand for $\phi[x := M]$, when P is $\{\, x \mid \phi \,\}$.

A proof reduces when an introducer is followed by the corresponding eliminator. We write $\pi[\phi := \rho]$ for the proof that results by substituting ρ for each use of (Id) on ϕ in the proof of π.

In addition to the listed reductions, we also have a number of commuting conversions for (β). Here is the commuting conversion to pass (β) down through ($\to$-E), on the assumption that $M = N$ is a substitution instance of an equation in β.

$$\dfrac{\dfrac{\Theta \vdash (\phi \to \psi)[x := M]}{\Theta \vdash (\phi \to \psi)[x := N]}\,\beta \qquad \Theta \vdash \phi[x := N]}{\Theta \vdash \psi[x := N]}\,{\to}\text{-E}$$

$$\Rightarrow \quad \dfrac{\Theta \vdash (\phi \to \psi)[x := M] \qquad \dfrac{\Theta \vdash \phi[x := N]}{\Theta \vdash \phi[x := M]}\,\beta}{\dfrac{\Theta \vdash \psi[x := M]}{\Theta \vdash \psi[x := N]}\,\beta}\,{\to}\text{-E}$$

There are similar conversions for each elimination rule. If an instance of β intervenes between an introduction rule and the corresponding elimination rule, preventing a reduction, then the commuting conversion can be used to push the rule beneath the elimination rule, allowing the reduction.

True, false, conjunction, and disjunction can be defined in terms of the connectives already given.

$$\begin{array}{lcl} \top & \equiv & \forall Z.\, () \in Z \to () \in Z \\ \bot & \equiv & \forall Z.\, () \in Z \\ \phi \wedge \psi & \equiv & \forall Z.\, (\phi \to \psi \to () \in Z) \to () \in Z \\ \phi \vee \psi & \equiv & \forall Z.\, (\phi \to () \in Z) \to (\psi \to () \in Z) \to () \in Z \end{array}$$

Here we assume Z does not appear free in ϕ or ψ, and write $()$ as shorthand for $\lambda x.\, x$. (It doesn't matter which term is chosen, any closed term works as well.)

As a further example of the power of P2, observe that it is powerful enough to express equality between terms. Following Leibniz, two terms are equal if one may be substituted for the other. Hence we define,

$$M = N \;\equiv\; \forall Z.\, M \in Z \to N \in Z.$$

That is, terms M and N are equal if any property Z that holds of M also holds of N. It is easy to see that equality is reflexive.

$$\begin{array}{ll} & M = M \\ \equiv & \quad \text{(definition)} \\ & \forall Z.\, M \in Z \to M \in Z \end{array}$$

It is more subtle to see that it is symmetric.

$$\begin{array}{ll} & M = N \\ \equiv & \quad \text{(definition)} \\ & \forall Z.\, M \in Z \to N \in Z \\ \Rightarrow & \quad (\text{instantiate } Z := \{\, x \mid x = M \,\}) \\ & M = M \to N = M \\ \Rightarrow & \quad \text{(equality is reflexive)} \\ & N = M \end{array}$$

One may similarly show transitivity, and that $M =_\beta N$ implies $M = N$.

A logic is *extensional* if whenever two terms are applied to equal arguments they return equal results. We define

$$(\forall x.\, M\, x = N\, x) \to M = N \qquad \text{(ext)}$$

where x does not appear free in M and N. We write P2 + ext for P2 with (ext) as an axiom. Extensionality for untyped terms is a stronger assumption than extensionality for typed terms. Arguably, it is stronger than one might wish, but several of our key proofs depend upon it.

3 The Reynolds embedding and the Girard projection

The Reynolds embedding takes a proof δ of a judgement

$$x_1 : A_1,\, \ldots,\, x_n : A_n \vdash u : B$$

in F2 into a proof δ^* of a judgement

$$x_1 \in A_1^*,\, \ldots,\, x_n \in A_n^* \vdash |u| \in B^*$$

in P2. Each type A maps into a predicate A^*, each typed lambda term t maps via type erasure into an untyped lambda term $|t|$, and each typing $t : A$ maps into the formula $|t| \in A^*$. The Reynolds embedding is defined in Figure 3.

The Girard projection takes a proof π of a judgement

$$\phi_1,\, \ldots,\, \phi_n \vdash \psi$$

in P2 into a proof π° of a judgement

$$z_1 : \phi_1^\circ,\, \ldots,\, z_n : \phi_n^\circ \vdash u : \psi^\circ$$

in F2. Here u is a typed term with free variables $z_1, \ldots, z_n$, which is uniquely determined by the proof π. The variables are assumed to be taken from a fixed list $z_1, z_2, \ldots$ of variables. Each formula ϕ maps into a type ϕ°. The Girard projection is defined in Figure 4.

We extend the Girard projection to predicates in the obvious way, taking $(\{\, x \mid \phi \,\})^\circ \equiv \phi^\circ$. Both the Reynolds embedding and the Girard projection

preserve substitution, so that $(A[X := B])^* \equiv A^*[X{:=}B^*]$ and $\phi[X{:=}\{\, x \mid \psi \,\}]^\circ \equiv \phi^\circ[X := \psi^\circ]$.

The Reynolds embedding and the Girard projection are homomorphisms, in that each preserves the reduction structure of derivations.

Proposition 1. *(Preservation of reductions) If* $\delta_0 \Rightarrow \delta_1$ *in* F2 *then* $\delta_0^* \Rightarrow \delta_1^*$ *in* P2*; and if* $\pi_0 \Rightarrow \pi_1$ *in* P2*, then* $\pi_0^\circ \Rightarrow \pi_1^\circ$ *in* F2.

Proof. It is easy to confirm that the Girard projection takes the $\to$ and $\forall^2$ reductions in P2 into the corresponding reductions of F2, and the $\forall^1$ reduction and commuting conversions of P2 into the identity. Similarly, the Reynolds embedding takes the $\to$ and $\forall^2$ reductions in F2 into the corresponding reductions of P2. A more subtle point is that the Reynolds embedding may introduce (β) rules that interfere with reductions, but these may always be pushed out of the way using commuting conversions. (Leivant's otherwise excellent tutorial [Lei90] appears to ignore role of commuting conversions.) □

We consider judgements and derivations in F2 as equivalent up to renaming of variables. That is, the judgements $x_1 : A_1, \ldots, x_n : A_n \vdash t : B$ and $y_1 : A_1, \ldots, y_n : A_n \vdash u : B$ are equivalent if $t = u[y_1 := x_1, \ldots, y_n := x_n]$.

The Reynolds embedding followed by the Girard projection is the identity.

Proposition 2. *(Girard inverts Reynolds) If* δ *is a derivation in* F2*, then* $\delta^{*\circ} \equiv \delta$*, up to renaming.*

Proof. Straightforward induction over the structure of derivations. □

As an example, here is the type of the Church numerals in F2.

$$\mathbf{Nat} \equiv \forall Z.\, (Z \to Z) \to Z \to Z$$

Applying the Reynolds embedding yields the following predicate in P2.

$$\mathbf{Nat}^* \;\equiv\; \{\, n \mid \forall Z.\, \forall s.\, (\forall m.\, m \in Z \to s\, m \in Z) \to \forall z.\, z \in Z \to n\, s\, z \in Z \,\}$$

It is easy to check that $\mathbf{Nat} = \mathbf{Nat}^{*\circ}$.

Let **two** be the second Church numeral in F2, and let two be its erasure, $\mathsf{two} \equiv |\mathbf{two}|$.

$$\begin{array}{rcl} \mathbf{two} & \equiv & \Lambda Z.\, \lambda s : Z \to Z.\, \lambda z : Z.\, s\, (s\, z) \\ \mathsf{two} & \equiv & \lambda s.\, \lambda z.\, s\, (s\, z) \end{array}$$

If δ_t is the derivation of $\vdash \mathbf{two} : \mathbf{Nat}$ in F2, then δ_t^* is the proof of $\vdash \mathsf{two} \in \mathbf{Nat}^*$ in P2. It is easy to check that $\delta_t \equiv \delta_t^{*\circ}$.

Note that the Girard projection takes equality $M = N$ into the unit type $\forall X.\, X \to X$. In the term model, or in any parametric model, the only value of this type is the identity function [Wad89]. Hence, the Girard projection erases any information content in the proof of an equality judgement.

Also observe that the erasure of the extensionality axiom (ext) has the type $(\forall X.\, X \to X) \to (\forall X.\, X \to X)$. One may extend the Girard projection so that it maps axiom (ext) into the derivation of the term $\lambda i : \forall X.\, X \to X.\, i$.

4 Doubling and Parametricity

The Reynolds embedding corresponds to a unary version of Reynolds's abstraction theorem. We can recover the binary version by means of a *doubling* mapping from P2 to P2.

We represent binary relations as predicates over pairs. We use the usual encoding of pairs in lambda calculus.

$$\begin{array}{lcl} (M, N) & \equiv & \lambda k.\, kMN \\ \mathsf{fst} & \equiv & \lambda z.\, z(\lambda x.\, \lambda y.\, x) \\ \mathsf{snd} & \equiv & \lambda z.\, z(\lambda x.\, \lambda y.\, y) \\ \{\, (x, y) \mid \phi \,\} & \equiv & \{\, z \mid \phi[x := \mathsf{fst}\ z,\ y := \mathsf{snd}\ z] \,\} \end{array}$$

Observe that $(M, N) \in \{\, (x, y) \mid \phi \,\}$ simplifies to $\phi[x := M,\ y := N]$ as required.

Doubling is defined with the aid of operations that rename variables. For each individual variable x there exists a renaming x'; and for each propositional variable X there exists a renaming $X^\ddagger$. We write M' for the term that results by renaming each free variable x in M to x'.

Doubling takes a proof δ of a judgement

$$\Theta \vdash \phi$$

in P2 into a proof $\delta^\ddagger$ of a judgement

$$\Theta^\ddagger \vdash \phi^\ddagger$$

in P2. Each formula ϕ maps into a formula $\phi^\ddagger$. Doubling is defined in Figure 5.

We extend doubling to predicates in the obvious way, taking $(\{\, x \mid \phi \,\})^\ddagger \equiv \{\, (M, M') \mid \phi^\ddagger \,\}$. Doubling preserves substitution, so that $\phi[Z := \psi]^\ddagger \equiv \phi^\ddagger[Z^\ddagger := \psi^\ddagger]$.

Again, doubling is a homomorphism.

Proposition 3. *(Doubling) If* $\pi_0 \Rightarrow \pi_1$ *in* P2 *then* $\pi_0^\ddagger \Rightarrow \pi_1^\ddagger$ *in* P2.

Proof. By examination of the reduction rules. □

What Reynolds calls the Abstraction Theorem [Rey83] and what Plotkin and Abadi call the Logical Relations Lemma [PA93] arises as the composition of the Reynolds embedding with doubling.

Proposition 4. *(Abstraction Theorem) If*

$$x_1 : A_1, \ldots, x_n : A_n \vdash t : B$$

is derivable in F2*, then*

$$(x_1, x_1') \in A_1^{*\ddagger}, \ldots, (x_n, x_n') \in A_n^{*\ddagger} \vdash (|t|, |t|') \in B^{*\ddagger}$$

is derivable in P2*.*

Proof. Immediate, since the Reynolds embedding followed by doubling takes a derivation of the antecedent into a derivation of the consequent. □

As an example, here is the type of the Church numerals in F2.

$$\mathbf{Nat} \equiv \forall Z.\, (Z \to Z) \to Z \to Z$$

Applying the Reynolds embedding followed by doubling yields the following predicate in P2.

$$\begin{array}{l}\mathbf{Nat}^{*\ddagger} \equiv \\ \quad \{\, (n, n') \mid \forall Z^{\ddagger}. \\ \qquad \forall s.\, \forall s'.\, (\forall m.\, \forall m'.\, (m, m') \in Z^{\ddagger} \to (s\, m, s'\, m') \in Z^{\ddagger}) \\ \qquad\quad \to \forall z.\, \forall z'.\, (z, z') \in Z^{\ddagger} \to (n\, s\, z, n'\, s'\, z') \in Z^{\ddagger} \,\}\end{array}$$

Similarly, starting with the derivation δ_t of the second Church numeral **two** in F2 and applying the Reynolds embedding followed by doubling yields the derivation $\delta_t^{*\ddagger}$ in P2 of the judgement

$$\vdash (\mathsf{two}, \mathsf{two}) \in \mathbf{Nat}^{*\ddagger}$$

Here renaming has no effect on the two terms, which contain only bound variables.

The identity relation on a type is equality restricted to that type.

Definition 5. *The* identity relation *on type A is written $A^{=}$.*

$$A^{=} \;\equiv\; \{\, (z, z') \mid z = z' \wedge z \in A^{*} \,\}$$

It is easy to verify that $A^{=}$ is reflexive, symmetric, and transitive, and that $(z, z') \in A^{=}$ implies $z \in A^{*}$ and $z' \in A^{*}$.

The parametric closure of a type is the doubling of the Reynolds embedding of that type, with the relation corresponding to each free type variable taken to be the identity relation.

Definition 6. *The* parametric closure *of type A is written $A^{\approx}$.*

$$A^{\approx} \;\equiv\; A^{*\ddagger}[X_1^{\ddagger} := X_1^{=}, \ldots, X_1^{\ddagger} := X_1^{=}]$$

Here $X_1, \ldots, X_n$ are the free type variables in A.

It is easy to verify that $A^{\approx}$ is symmetric and transitive, so it is a partial equivalence relation, and that $(z, z') \in A^{\approx}$ implies that $z \in A^{*}$ and $z' \in A^{*}$.

It is interesting to consider those cases where the identity relation implies the parametric closure, or conversely.

Definition 7. *We say that type A is* parametric *when $A^{=} \subseteq A^{\approx}$, and* extensive *when $A^{\approx} \subseteq A^{=}$.*

An assertion that every type is parametric corresponds to Reynolds's parametricity condition, and an assertion that every type is parametric and extensive might correspond to Reynolds's Identity Extension Lemma [Rey83].

In Section 5, we will see that all algebraic types are parametric and extensive, in any model for which algebraic types have the usual inductive properties.

Is it consistent to assume that every type is parametric? This conjecture is plausible, since for any closed term M and closed type A, if $M \in A^*$ is provable, then by doubling $(M, M) \in A^{\approx}$ is also provable. But as of this writing neither a proof nor a counterexample has been found. We do not pursue the point further in this paper.

However, it would not be consistent to assume that every type is both parametric and extensive. Assume that types A and B are parametric and extensive and that $f, g \in (A \to B)^*$. If $A \to B$ is extensive, it follows that $f\,x = g\,x$ for all $x \in A^*$ implies $f = g$. But this is not appropriate, since in the untyped world we may apply f and g to arguments that are not of type A.

It is easy to construct a counter-example demonstrating this problem. (Easy, but not obvious; I'm grateful to an anonymous referee for the following.) As usual, let $\mathsf{zero} \equiv \lambda s.\,\lambda z.\,z$, $\mathsf{I} \equiv \lambda x.\,x$, $\mathsf{K} \equiv \lambda x.\,\lambda y.\,x$, and take $f \equiv \lambda n.\,\mathsf{zero}$ and $g \equiv \lambda n.\,n\,\mathsf{I}\,\mathsf{zero}$. We have $f\,n = \mathsf{zero} = g\,n$ for every $n \in \mathbf{Nat}^*$. (The second equality is easily proved by induction, which is justified by the results in Section 5 on the assumption that Nat is parametric.) Clearly, $f, g \in \mathrm{Nat} \to \mathrm{Nat}$, so if $\mathrm{Nat} \to \mathrm{Nat}$ is extensive we would conclude $f = g$. But this is false, since $f\,\mathsf{K} = \mathsf{zero} \neq \mathsf{I} = g\,\mathsf{K}$.

These considerations suggest that $A^{=}$ is not, in general, an appropriate definition of equality at type A in this framework. Indeed, it may be appropriate to simply take $A^{\approx}$ as the definition of equality at type A, in which case the Identity Extension Lemma holds by definition.

5 Algebraic types and parametricity

The grammar of algebraic types is shown in Figure 6. An algebraic type is either a type variable Z, the natural number type Nat, the empty type 0, the sum type $T + U$, the unit type 1, the product type $T \times U$, or the fixpoint type $\mu Z.\,T$. The natural number type is included only for illustration, since it is equivalent to the type $\mu Z.\,Z + 1$. The subtypes of a type are defined as usual, so for instance the subtypes of $\mu Z.\,Z + 1$ are Z, 1, $Z + 1$, and $\mu Z.\,Z + 1$ itself.

Each algebraic type has two interpretations, an *inductive* interpretation as a predicate in P2, and a *deductive* interpretation as a type in F2.

The inductive interpretation defines each type by its corresponding induction principle. For instance, $\mathsf{Nat} = \lceil \mathrm{Nat} \rceil$ is the induction principle for natural numbers.

$$\mathsf{Nat} \equiv \{\, n \mid \forall Z.\,(\forall m.\,m \in Z \to \mathsf{succ}\,m \in Z) \to \mathsf{zero} \in Z \to n \in Z \,\}$$

To prove a property of natural numbers by induction, one must show that for all m if m has the property then $\mathsf{succ}\,m$ has the property, and one must show

that zero has the property. The above definition states that a value is a natural number if one can prove a property of it by induction. The idea of classifying induction principles using second-order propositional variables, and of defining a type via its induction principle, goes back to Frege [Fre79].

One immediate consequence of the definition is that succ and zero do indeed construct natural numbers. Similar properties follow for all the algebraic types.

Proposition 8. *(Constructor Lemma) The following are provable in* P2*.*

$$\begin{array}{lll}
n \in \mathsf{Nat} & \to \mathsf{succ}\, n & \in \mathsf{Nat} \\
 & \mathsf{zero} & \in \mathsf{Nat} \\
x \in \lceil T \rceil & \to \mathsf{inl}\, x & \in \lceil T + U \rceil \\
y \in \lceil U \rceil & \to \mathsf{inr}\, y & \in \lceil T + U \rceil \\
 & \mathsf{unit} & \in \lceil 1 \rceil \\
x \in \lceil T \rceil \wedge y \in \lceil U \rceil & \to \mathsf{pair}\, x\, y & \in \lceil T \times U \rceil \\
x \in \lceil T[Z := \mu Z.\, T] \rceil & \to \mathsf{in}\, x & \in \lceil \mu Z.\, T \rceil
\end{array}$$

Proof. Straightforward. □

The inductive definitions, and the above proposition, do not depend on the particular value of the terms chosen for the constructors succ and zero and so on. They might, for example, be chosen to be uninterpreted constants. However it turns out there is a good reason to choose them to be particular untyped lambda terms, namely the erasures of the typed lambda terms that are the corresponding constructors in the deductive interpretation.

The deductive interpretation maps each type to the usual definition in second order lambda calculus. For example, the deductive interpretation $\mathbf{Nat} \equiv \lfloor \mathrm{Nat} \rfloor$ is the usual Church type of the naturals. Corresponding to this definiton are the usual constructors for the natural numbers in second order lambda calculus, namely $\mathbf{succ} : \mathbf{Nat} \to \mathbf{Nat}$ and $\mathbf{zero} : \mathbf{Nat}$.

The other definitions for each type and each constructor are entirely standard. To define the constructor for fixpoints, one makes use of the fact that if T is an algebraic type in which Z is free, then Z appears positively in the deductive translation $\lfloor T \rfloor$, and that if A is a polymorphic type in which Z appears positively, then for each $f : B \to C$ there is a canonically defined function $A[f] : A[Z{:=}B] \to A[Z{:=}C]$. (For details of the construction, see, e.g., Reynolds and Plotkin [RP90].)

Both interpretations preserve substitution, so that $\lceil T[Z := U] \rceil = \lceil T \rceil\, [Z := \lceil U \rceil]$ and $\lfloor T[Z := U] \rfloor = \lfloor T \rfloor\, [Z := \lfloor U \rfloor]$.

The deductive interpretation is so named because it can be deduced by applying the Girard projection to the inductive interpretation.

Proposition 9. *(Girard projection takes inductive to deductive) For every algebraic type T, we have $\lfloor T \rfloor \equiv \lceil T \rceil^{\circ}$.*

Proof. Obvious by inspection. □

Furthermore, the constructors in the deductive interpretation can be deduced from the constructors of the inductive interpretation. For instance, if succ and zero are uninterpreted constants, then there are canonical proofs in P2 of the following propositions; call these π_s and π_z.

$$\vdash n \in \mathsf{Nat} \to \mathsf{succ}\, n \in \mathsf{Nat} \quad \text{and} \quad \vdash \mathsf{zero} \in \mathsf{Nat}.$$

Applying the Girard projection to these proofs yields derivations in F2 of the following typings; call these $\delta_s \equiv \pi_s^\circ$ and $\delta_z \equiv \pi_z^\circ$.

$$\vdash \mathbf{succ} : \mathbf{Nat} \to \mathbf{Nat} \quad \text{and} \quad \vdash \mathbf{zero} : \mathbf{Nat}$$

In what follows, we assume that each of the inductive constructors is the erasure of the corresponding deductive constructor. That is, $\mathsf{succ} = |\mathbf{succ}|$ and $\mathsf{zero} = |\mathbf{zero}|$, and so on for the constructors of the other algebraic types. With this assumption, we will be able to show not only that the Girard projection takes the inductive interpretation into the deductive one, but also that, in the presence of parametricity, the Reynolds embedding takes the deductive interpretation into the inductive one.

Definition 10. *We say that algebraic type T is* deductive *when $\lceil T \rceil \subseteq \lfloor T \rfloor^*$ and* inductive *when $\lfloor T \rfloor^* \subseteq \lceil T \rceil$.*

We will show that all algebraic types are deductive. Further, for algebraic types, parametricity and inductiveness are equivalent: if all subtypes of a given algebraic type are parametric then that type is inductive, and if all subtypes of a given type are inductive then that type is parametric.

Thus, for algebraic types in the presence of parametricity, the inductive and deductive definitions are equivalent. That is, one has not only $\lfloor T \rfloor \equiv \lceil T \rceil^\circ$ but also $\lceil T \rceil = \lfloor T \rfloor^*$, and hence not only $\lfloor T \rfloor \equiv \lfloor T \rfloor^{*\circ}$ but also $\lceil T \rceil = \lceil T \rceil^{\circ *}$. In particular, we have $\mathbf{Nat} \equiv \mathsf{Nat}^\circ$ and $\mathsf{Nat} = \mathbf{Nat}^*$. Thus, there is a sense in which the Reynolds and Girard translations are not merely an embedding-projection pair, but truly inverses.

5.1 Every algebraic type is extensive

As promised in Section 4, we show that every algebraic type is extensive, which corresponds, in a rough sense, to one half of Reynolds's Identity Extension Lemma [Rey83]. We require extensionality.

Proposition 11. *(Every algebraic type is extensive) If T is an algebraic type, then* P2 + ext *proves $\lfloor T \rfloor^{\approx} \subseteq \lfloor T \rfloor^{=}$.*

Proof. The case for a free type variable X is immediate, since $X^{\approx} = X^{=}$ by definition. Below is the proof for $T \times U$. The other cases are similar.

$$\begin{aligned}
& (z, z') \in \lfloor T \times U \rfloor^{\approx} \\
= \quad & \text{(definition parametric closure)} \\
& \forall Z^{\ddagger}.\, \forall k, k'. \\
& \quad (\forall x, x'.\, (x, x') \in \lfloor T \rfloor^{\approx} \to \forall y, y'.\, (y, y') \in \lfloor U \rfloor^{\approx} \to (k\, x\, y, k'\, x'\, y') \in Z^{\ddagger}) \\
& \quad \to (z\, k, z'\, k') \in Z^{\ddagger}
\end{aligned}$$

$\Rightarrow$ (instantiate $Z^\ddagger := \{\,(z, z') \mid z = z'\,\}$)
$\forall k, k'.$
$(\forall x, x'.\,(x, x') \in \lfloor T \rfloor^{\approx} \to \forall y, y'.\,(y, y') \in \lfloor U \rfloor^{\approx} \to k\ x\ y = k'\ x'\ y')$
$\to z\ k = z'\ k'$
$\Rightarrow$ (induction hypothesis)
$\forall k, k'.\,(\forall x, x'.\,x = x' \to \forall y, y'.\,y = y' \to k\ x\ y = k'\ x'\ y') \to z\ k = z'\ k'$
$\Rightarrow$ (extensionality on k and k')
$\forall k, k'.\,k = k' \to z\ k = z'\ k'$
$\Rightarrow$ (extensionality on z and z')
$z = z'$

□

5.2 Every algebraic type is deductive

We show that every algebraic type is deductive. We do not require extensionality.

Proposition 12. *(Every algebraic type is deductive) If T is an algebraic type, then* P2 *proves $\lceil T \rceil \subseteq \lfloor T \rfloor^*$.*

Proof. Below is the proof for $T \times U$. The other cases are similar.

$z \in \lceil T \times U \rceil$
$\equiv$ (definition inductive interpretation)
$\forall Z.\,(\forall x.\,x \in \lceil T \rceil \to \forall y.\,y \in \lceil U \rceil \to \mathsf{pair}\ x\ y \in Z) \to z \in Z$
$\Rightarrow$ (take $Z = \lfloor T \times U \rfloor^*$)
$(\forall x.\,x \in \lceil T \rceil \to \forall y.\,y \in \lceil U \rceil \to \mathsf{pair}\ x\ y \in \lfloor T \times U \rfloor^*) \to z \in \lfloor T \times U \rfloor^*$
$\Rightarrow$ (induction hypothesis)
$(\forall x.\,x \in \lfloor T \rfloor^* \to \forall y.\,y \in \lfloor U \rfloor^* \to \mathsf{pair}\ x\ y \in \lfloor T \times U \rfloor^*) \to z \in \lfloor T \times U \rfloor^*$
$\Rightarrow$ (Reynolds embedding applied to **pair**)
$z \in \lfloor T \times U \rfloor^*$

□

5.3 Inductive implies parametric

If all algebraic types are inductive then they are also parametric. Extensionality is not required. Inductiveness is required not only of a type but also of all its subtypes, so that the induction hypothesis will apply.

Proposition 13. *(Inductive implies parametric) Let T be an algebraic type. If* P2 *proves $\lfloor S \rfloor^* \subseteq \lceil S \rceil$ for every subtype S of T, then* P2 *proves $\lfloor T \rfloor^{=} \subseteq \lfloor T \rfloor^{\approx}$.*

Proof. Below is the proof for $T \times U$. The other cases are similar.

$z \in \lfloor T \times U \rfloor^*$
$\Rightarrow$ (by assumption $T \times U$ is inductive)
$z \in \lceil T \times U \rceil$
$\equiv$ (definition inductive interpretation)
$\forall Z.\,(\forall x.\,x \in \lceil T \rceil \to \forall y.\,y \in \lceil U \rceil \to \mathsf{pair}\ x\ y \in Z) \to z \in Z$
$\Rightarrow$ (instantiate $Z := \{\,z \mid (z, z) \in \lfloor T \times U \rfloor^{\approx}\,\}$)

$(\forall x.\, x \in \lceil T \rceil \to \forall y.\, y \in \lceil U \rceil \to (\mathsf{pair}\ x\ y, \mathsf{pair}\ x\ y) \in \lfloor T \times U \rfloor^{\approx})$
$\to (z, z) \in \lfloor T \times U \rfloor^{\approx}$
$\Rightarrow$ (Proposition 12 and induction hypothesis)
$(\forall x.\, (x, x) \in \lfloor T \rfloor^{\approx} \to \forall y.\, (y, y) \in \lfloor U \rfloor^{\approx} \to (\mathsf{pair}\ x\ y, \mathsf{pair}\ x\ y) \in \lfloor T \times U \rfloor^{\approx})$
$\to (z, z) \in \lfloor T \times U \rfloor^{\approx}$
$\Rightarrow$ (Abstraction Theorem applied to **pair**)
$(z, z) \in \lfloor T \times U \rfloor^{\approx}$

□

Reynolds and Plotkin [RP90] were the first to suggest that parametricity implies inductivity, and so far as I know, Hasegawa [Has94] was the first to suggest the converse. (Hasegawa asserts that a type is inductive if it is parametric, while here we assert that a type is inductive if all its subtypes are parametric. Analysis of Hasegawa's proof shows that he seems to have inadvertently omitted the condition on subtypes, since he relies on the Identity Extension Lemma at the subtypes and this depends on parametricity.)

5.4 Parametric implies inductive

Every algebraic type that is parametric is also inductive. We require extensionality. Parametricity is required not only of a type but also of all its subtypes, so that the induction hypothesis will apply.

Proposition 14. *(Parametric implies inductive) Let T be an algebraic type. If* $\mathrm{P}2 + \mathrm{ext}$ *proves* $\lfloor S \rfloor^{=} \subseteq \lfloor S \rfloor^{\approx}$ *for every subtype S of T, then* $\mathrm{P}2 + \mathrm{ext}$ *proves* $\lfloor T \rfloor^{*} \subseteq \lceil T \rceil$.

The proof depends on the following lemma. Böhm and Berarducci [BB85, Theorem 7.3] prove a similar result, though in a different framework and with a different technique.

Proposition 15. *(Böhm's Lemma) The following are provable in* $\mathrm{P}2 + \mathrm{ext}$.

$$\begin{array}{lll}
(z, z) \in \lfloor \mathrm{Nat} \rfloor^{\approx} & \to z\ \mathsf{succ}\ \mathsf{zero} & = z \\
(z, z) \in \lfloor 0 \rfloor^{\approx} & \to z & = z \\
(z, z) \in \lfloor T + U \rfloor^{\approx} & \to z\ \mathsf{inl}\ \mathsf{inr} & = z \\
(z, z) \in \lfloor 1 \rfloor^{\approx} & \to z\ \mathsf{unit} & = z \\
(z, z) \in \lfloor T \times U \rfloor^{\approx} & \to z\ \mathsf{pair} & = z \\
(z, z) \in \lfloor \mu Z.\, T \rfloor^{\approx} & \to z\ \mathsf{in} & = z
\end{array}$$

Proof. Below is the proof for $T \times U$. The other cases are similar.

$z \in \lfloor T \times U \rfloor^{\approx}$
$\Rightarrow$ (definition deductive interpretation, doubling)
$\forall Z^{\ddagger}.\, \forall k, k'.$
$(\forall x, x'.\, (x, x') \in \lfloor T \rfloor^{\approx} \to \forall y, y'.\, (y, y') \in \lfloor U \rfloor^{\approx} \to (k\ x\ y, k'\ x'\ y') \in Z^{\ddagger})$
$\to (z\ k, z\ k') \in Z^{\ddagger}$

$\Rightarrow$ (instantiate $Z^{\ddagger} := \{\, (z, z') \mid z\, k'' = z' \,\}$ and $k := \mathsf{pair}$)
$\forall k', k''.$
$(\forall x, x'.\, (x, x') \in \lfloor T \rfloor^{\approx} \to \forall y, y'.\, (y, y') \in \lfloor U \rfloor^{\approx} \to \mathsf{pair}\; x\; y\; k'' = k'\; x'\; y')$
$\to z\; \mathsf{pair}\; k'' = z\; k')$
$\Rightarrow$ (T and U are extensive, β reduction on pair)
$\forall k', k''.$
$(\forall x, x'.\, x = x' \to \forall y, y'.\, y = y' \to k''\; x\; y = k'\; x'\; y')$
$\to z\; \mathsf{pair}\; k'' = z\; k')$
$\Rightarrow$ (extensionality on k'' and k')
$\forall k', k''.\, k'' = k' \to z\; \mathsf{pair}\; k'' = z\; k')$
$\Rightarrow$ (extensionality on $z\; \mathsf{pair}$ and z)
$z\; \mathsf{pair} = z$

□

We can now prove Proposition 14.

Proof. Below is the proof for $T \times U$. The other cases are similar.

$z \in \lfloor T \times U \rfloor^{*}$
$\Rightarrow$ (definition deductive interpretation, Reynolds embedding)
$\forall Z.\, \forall k.\, (\forall x.\, x \in \lfloor T \rfloor^{*} \to \forall y.\, y \in \lfloor U \rfloor^{*} \to k\; x\; y \in Z) \to z\; k \in Z$
$\Rightarrow$ (instantiate $Z := \lceil T \times U \rceil$ and $k := \mathsf{pair}$)
$(\forall x.\, x \in \lfloor T \rfloor^{*} \to \forall y.\, y \in \lfloor U \rfloor^{*} \to \mathsf{pair}\; x\; y \in \lceil T \times U \rceil) \to z\; \mathsf{pair} \in \lceil T \times U \rceil$
$\Rightarrow$ (induction hypothesis)
$(\forall x.\, x \in \lceil T \rceil \to \forall y.\, y \in \lceil U \rceil \to \mathsf{pair}\; x\; y \in \lceil T \times U \rceil) \to z\; \mathsf{pair} \in \lceil T \times U \rceil$
$\Rightarrow$ (Constructor Lemma)
$z\; \mathsf{pair} \in \lceil T \times U \rceil$
$=$ (parametricity ($z \in \lfloor T \times U \rfloor^{*} \to (z, z) \in \lfloor T \times U \rfloor^{\approx}$), Böhm's Lemma)
$z \in \lceil T \times U \rceil$

□

An immediate corollary is the following.

Proposition 16. *(Girard-Reynolds isomorphism) If T is an algebraic type then $\lfloor T \rfloor \equiv \lfloor T \rfloor^{*\circ}$, and furthermore if T and its subtypes are parametric then* P2 + ext *proves $\lceil T \rceil = \lceil T \rceil^{\circ *}$.*

Proof. Immediate from Propositions 9, 12 and 14. □

Acknowledgements Thanks to Andrew Pitts, Jon Riecke, and referees of POPL 1999, LICS 2000, and TACS 2001 for comments on this work; with special thanks to an anonymous referee and to Andrew Pitts for spotting errors.

References

[ACC93] M. Abadi, L. Cardelli, and P.-L. Curien, Formal Parametric Polymorphism, *Theoretical Computer Science* 121(1–2):9–58, December 1993. (Part of A Collection of Contributions in Honour of Corrado Boehm on the Occasion of his 70th Birthday.) Also appeared as SRC Research Report 109.

[Bar91] H. Barendregt, Introduction to generalized types systems, *Journal of Functional Programming*, 1(2):125–154, April 1991.

[BFSS90] E. S. Bainbridge, P. J. Freyd, A. Scedrov, and P. J. Scott, Functorial polymorphism, in G. Huet, editor, *Logical Foundations of Functional Programming*, pp. 315–330, Addison-Wesley, 1990.

[BB85] C. Böhm and A. Berarducci, Automatic synthesis of typed Λ-programs on term algebras, *Theoretical Computer Science* 39(2–3):135–154, August 1985.

[BC88] V. Breazu-Tannen and T. Coquand, Extensional models for polymorphism, *Theoretical Computer Science*, 59:85–114, 1988.

[CF58] H. B. Curry and R. Feys, *Combinatory Logic*, North Holland, 1958.

[CH88] T. Coquand and G. Huet, The calculus of constructions, *Information and Computation*, 76:95–120, 1988.

[Con86] R. Constable, *et al.*, *Implementing mathematics with the Nuprl proof development system*, Prentice-Hall, 1986.

[deB70] N. G. de Bruijn, The mathematical language of AUTOMATH, its usage and some of its extensions, *Proceedings of the Symposium on Automatic Demonstration*, LNCS 125, Springer-Verlag, 1970.

[DM82] L. Damas and R. Milner, Principal type schemes for functional programs, *9'th Annual Symposium on Principles of Programming Languages*, Albuquerque, N.M., January 1982.

[Fre79] Gottlob Frege. Begriffsschrift, a formula language, modeled upon that of arithmetic, for pure thought (1879). In Jan van Heijenoort, editor, *From Frege to Gödel: A Source Book in Mathematical Logic, 1879-1931.* Harvard University Press, 1967.

[Gir72] J.-Y. Girard, *Interprétation functionelle et élimination des coupures dans l'arithmétique d'ordre supérieure*, Ph.D. thesis, Université Paris VII, 1972.

[GLT89] J.-Y. Girard, Y. Lafont, and P. Taylor, *Proofs and Types*, Cambridge University Press, 1989.

[Has94] R. Hasegawa, Categorical data types in parametric polymorphism, *Mathematical Structures in Computer Science*, 4:71-109, (1994).

[Hin69] R. Hindley, The principal type scheme of an object in combinatory logic, *Trans. Am. Math. Soc.*, 146:29–60, December 1969.

[How80] W. A. Howard, The formulae-as-types notion of construction, in J. P. Seldin and J. R. Hindley, editors, *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus, and Formalism*, Academic Press, 1980. (The original version was circulated privately in 1969.)

[KP90] J.-L. Krivine and M. Parigot, Programming with proofs, *J. Inf. Process. Cybern. (EIK)*, 26(3): 149–167, 1990. (Revised version of a lecture presented at the 6'th International Symposium on Computation Theory, (SCT '87), Wednisch-Rietz, GDR, 30 November–4 December 1987.

[Lei83] D. Leivant, Reasoning about functional programs and complexity classes associated with type disciplines, *24'th Symposium on Foundations of Computer Science*, Washington D.C., IEEE, 460–469.

[Lei90] D. Leivant, Contracting proofs to programs, in P. Odifreddi, editor, *Logic and Computer Science*, Academic Press, 1990.

[Mai91] H. Mairson, Outline of a proof theory of parametricity, in J. Hughes, editor, *5'th International Conference on Functional Programming Languages and Computer Architecture*, Springer-Verlag LNCS 523, Cambridge, Massachusetts, August 1991.

[Mar82] P. Martin-Löf, Constructive mathematics and computer programming, *6'th International Congress for Logic, Methodology, and Philosophy of Science*, North Holland, pp. 153–175, 1982.

[Mil78] R. Milner, A theory of type polymorphism in programming, *Journal of Computers and Systems Science*, 17:348–375, 1978.

[Mog86] E. Moggi, Communication to the Types electronic forum, 10 February 1986.

[MR91] Q. Ma and J. C. Reynolds, Types, abstraction, and parametric polymorphism, part 2, in S. Brookes *et al.* editors, *Mathematical Foundations of Programming Semantics*, Springer Verlag LNCS, 1991.

[Pit87] A. M. Pitts, Polymorphism Is Set Theoretic, Constructively, in D. H. Pitt and A. Poigné and D. E. Rydeheard, editors, *Category Theory and Computer Science*, pages 12–39, Edinburgh 1987.

[Pit89] A. M. Pitts, Non-trivial power types can't be subtypes of polymorphic types, in *4th Annual Symposium on Logic in Computer Science*, pages 6–13, IEEE Computer Society Press, Washington 1989.

[Pit98] A. M. Pitts, Parametric polymorphism and operational equivalence, technical report 453, Cambridge University Computer Laboratory, 1998.

[PA93] G. Plotkin and M. Abadi, A logic for parametric polymorphism, in M. Bezem and J. F. Groote, editors, *Typed Lambda Calculi and Applications*, LNCS 664, Springer-Verlag, pp. 361–375, March 1993.

[PAC94] G. Plotkin, M. Abadi, and L. Cardelli, Subtyping and parametricity, *9'th Annual Symposium on Logic in Computer Science*, pp. 310–319, July 1994.

[Rey74] J. C. Reynolds, Towards a theory of type structure, in B. Robinet, editor, *Colloque sur la Programmation*, LNCS 19, Springer-Verlag.

[Rey83] J. C. Reynolds, Types, abstraction, and parametric polymorphism, in R. E. A. Mason, editor, *Information Processing 83*, pp. 513–523, North-Holland, Amsterdam, 1983.

[Rey84] J. C. Reynolds, Polymorphism is not set theoretic, in Kahn, MacQueen, and Plotkin, editors, *Semantics of Data Types*, Sophia-Antipolis, France, pp. 145–156, LNCS 173, Springer-Verlag, 1984.

[Rey90] J. C. Reynolds, Introduction to Part II: Polymorphic Lambda Calculus, in G. Huet, editor, *Logical Foundations of Functional Programming*, Addison-Wesley, 1990.

[RP90] J. C. Reynolds and G. D. Plotkin, On Functors Expressible in the polymorphic typed lambda calculus, in G. Huet, editor, *Logical Foundations of Functional Programming*, pp. 127–152, Addison-Wesley, 1990.

[Str67] C. Strachey, Fundamental concepts in programming languages, Lecture notes, International Summer School in Computer Programming, Copenhagen, August 1967. Reprinted in *Higher-Order and Symbolic Computation* 13(1/2):11-49, May 2000.

[Tak98] Izumi Takeuiti, An axiomatic system of parametricity, *Fundamenta Informaticae*, 33:397-432, IOS Press, 1998.

[Wad89] P. Wadler, Theorems for free!, *4'th International Conference on Functional Programming Languages and Computer Architecture*, ACM Press, London, September 1989.

[Wad91] P. Wadler, Recursive types for free!, manuscript, 1991.

Syntax

Type variables	X, Y, Z	
Individual variables	x, y, z	
Types	A, B	$::= X \mid A \to B \mid \forall X.\, B$
Typed terms	s, t, u	$::= x \mid \lambda x{:}A.\, u \mid s\ t \mid \Lambda X.\, u \mid s\ A$
Contexts	Γ	$::= x_1 : A_1, \ldots, x_n : A_n$

Rules

$$\frac{}{x_1 : A_1, \ldots, x_n : A_n \vdash x_i : A_i}\ \mathrm{Id}$$

$$\frac{\Gamma, x : A \vdash u : B}{\Gamma \vdash \lambda x{:}A.\, u : A \to B}\ {\to}\text{-I} \qquad \frac{\Gamma \vdash s : A \to B \qquad \Gamma \vdash t : A}{\Gamma \vdash s\ t : B}\ {\to}\text{-E}$$

$$\frac{\Gamma \vdash u : B}{\Gamma \vdash \Lambda X.\, u : \forall X.\, B}\ \forall^2\text{-I} \quad \text{when } X \text{ not free in } \Gamma \qquad \frac{\Gamma \vdash s : \forall X.\, B}{\Gamma \vdash s\ A : B[X := A]}\ \forall^2\text{-E}$$

Reductions

$$\frac{\dfrac{\overset{\vdots\,\delta}{\Gamma, x : A \vdash u : B}}{\Gamma \vdash \lambda x{:}A.\, u : A \to B}\ {\to}\text{-I} \qquad \overset{\vdots\,\epsilon}{\Gamma \vdash t : A}}{\Gamma \vdash (\lambda x{:}A.\, u)\ t : B}\ {\to}\text{-E} \quad \Rightarrow \quad \overset{\vdots\,\delta[x := \epsilon]}{\Gamma \vdash u[x := t] : B}$$

$$\frac{\dfrac{\overset{\vdots\,\delta}{\Gamma \vdash u : B}}{\Gamma \vdash \Lambda X.\, u : \forall X.\, B}\ \forall^2\text{-I}}{\Gamma \vdash (\Lambda X.\, u)\ A : B[X := A]}\ \forall^2\text{-E} \quad \Rightarrow \quad \overset{\vdots\,\delta[X := A]}{\Gamma \vdash u[X := A] : B[X := A]}$$

Fig. 1. Second-order lambda calculus (F2)

Syntax

Predicate variables	X, Y, Z	
Individual variables	x, y, z	
Terms	L, M, N	$::= x \mid \lambda x.\, M \mid M\ N$
Formulas	ϕ, ψ	$::= M \in X \mid \phi \to \psi \mid \forall x.\, \psi \mid \forall X.\, \psi$
Contexts	Θ	$::= \phi_1, \ldots, \phi_n$

Rules

$$\frac{}{\phi_1, \ldots, \phi_n \vdash \phi_i}\ \mathrm{Id}$$

$$\frac{\Theta, \phi \vdash \psi}{\Theta \vdash \phi \to \psi}\ {\to}\text{-I} \qquad \frac{\Theta \vdash \phi \to \psi \qquad \Theta \vdash \phi}{\Theta \vdash \psi}\ {\to}\text{-E}$$

$$\frac{\Theta \vdash \psi}{\Theta \vdash \forall x.\, \psi}\ \forall^1\text{-I} \quad \text{when } x \text{ not free in } \Theta \qquad \frac{\Theta \vdash \forall x.\, \psi}{\Theta \vdash \psi[x := M]}\ \forall^1\text{-E}$$

$$\frac{\Theta \vdash \psi}{\Theta \vdash \forall X.\, \psi}\ \forall^2\text{-I} \quad \text{when } X \text{ not free in } \Theta \qquad \frac{\Theta \vdash \forall X.\, \psi}{\Theta \vdash \psi[X := \{\, x \mid \phi \,\}]}\ \forall^2\text{-E}$$

$$\frac{(\Theta \vdash \psi)\,[x := M]}{(\Theta \vdash \psi)\,[x := N]}\ \beta \quad \text{when } M =_\beta N$$

Reductions

$$\frac{\dfrac{\overset{\vdots\ \pi}{\Theta, \phi \vdash \psi}}{\Theta \vdash \phi \to \psi}\ {\to}\text{-I} \qquad \overset{\vdots\ \rho}{\Theta \vdash \phi}}{\Theta \vdash \psi}\ {\to}\text{-E} \quad \Rightarrow \overset{\vdots\ \pi[\phi := \rho]}{\Theta \vdash \psi}$$

$$\frac{\dfrac{\overset{\vdots\ \pi}{\Theta \vdash \psi}}{\Theta \vdash \forall x.\, \psi}\ \forall\text{-I}}{\Theta \vdash \psi[x := M]}\ \forall\text{-E} \quad \Rightarrow \overset{\vdots\ \pi[x := M]}{\Theta \vdash \psi[x := M]}$$

$$\frac{\dfrac{\overset{\vdots\ \pi}{\Theta \vdash \psi}}{\Theta \vdash \forall X.\, \psi}\ \forall^2\text{-I}}{\Theta \vdash \psi[X := \{\, x \mid \phi \,\}]}\ \forall^2\text{-E} \quad \Rightarrow \overset{\vdots\ \pi[X := \{\, x \mid \phi \,\}]}{\Theta \vdash \psi[X := \{\, x \mid \phi \,\}]}$$

Fig. 2. Second-order propositional logic (P2)

Terms

$$\begin{array}{lcl} |x| & \equiv & x \\ |\lambda x{:}A.\, u| & \equiv & \lambda x.\, |u| \\ |s\ t| & \equiv & |s|\ |t| \\ |\Lambda X.\, u| & \equiv & |u| \\ |s\ A| & \equiv & |s| \end{array}$$

Types

$$\begin{array}{lcl} X^* & \equiv & \{\ z \mid z \in X\ \} \\ (A \to B)^* & \equiv & \{\ z \mid \forall x.\, x \in A^* \to z\ x \in B^*\ \} \\ (\forall X.\, B)^* & \equiv & \{\ z \mid \forall X.\, z \in B^*\ \} \end{array}$$

Contexts

$$(x_1 : A_1,\ \ldots,\ x_n : A_n)^* \ \equiv\ x_1 \in A_1^*,\ \ldots,\ x_n \in A_n^*$$

Judgements

$$(\Gamma \vdash t : A)^* \ \equiv\ \Gamma^* \vdash |t| \in A^*$$

Proofs

$$\left(\frac{}{x_1 : A_1,\ \ldots,\ x_n : A_n \vdash x_i : A_i}\ \mathrm{Id}\right)^* \quad \equiv \quad \frac{}{x_1 \in A_1^*,\ \ldots,\ x_n \in A_n^* \vdash x_i \in A_i^*}\ \mathrm{Id}$$

$$\left(\frac{\overset{\vdots\ \delta}{\Gamma,\ x : A \vdash u : B}}{\Gamma \vdash \lambda x.\, u : A \to B}\ {\to}\text{-I}\right)^* \quad \equiv \quad \frac{\dfrac{\dfrac{\overset{\vdots\ \delta^*}{\Gamma^*,\ x \in A^* \vdash |u| \in B^*}}{\Gamma^*,\ x \in A^* \vdash (\lambda x.\, |u|)\ x \in B^*}\ \beta}{\Gamma^* \vdash x \in A^* \to (\lambda x.\, |u|)\ x \in B^*}\ {\to}\text{-I}}{\Gamma^* \vdash \forall x.\, x \in A^* \to (\lambda x.\, |u|)\ x \in B^*}\ \forall^1\text{-I}$$

$$\left(\frac{\overset{\vdots\ \delta}{\Gamma \vdash s : A \to B} \qquad \overset{\vdots\ \epsilon}{\Gamma \vdash t : A}}{\Gamma \vdash s\ t : B}\ {\to}\text{-E}\right)^* \equiv \frac{\dfrac{\overset{\vdots\ \delta^*}{\Gamma^* \vdash \forall x.\, x \in A^* \to |s|\ x \in B^*}}{\Gamma^* \vdash |t| \in A^* \to |s|\ |t| \in B^*}\ \forall^1\text{-E} \qquad \overset{\vdots\ \epsilon^*}{\Gamma^* \vdash |t| \in A^*}}{\Gamma^* \vdash |s|\ |t| \in B^*}\ {\to}\text{-E}$$

$$\left(\frac{\overset{\vdots\ \delta}{\Gamma \vdash u : B}}{\Gamma \vdash \Lambda X.\, u : \forall X.\, B}\ \forall^2\text{-I}\right)^* \quad \equiv \quad \frac{\overset{\vdots\ \delta^*}{\Gamma^* \vdash |u| \in B^*}}{\Gamma^* \vdash \forall X.\, |u| \in B^*}\ \forall^2\text{-I}$$

$$\left(\frac{\overset{\vdots\ \delta}{\Gamma \vdash s : \forall X.\, B}}{\Gamma \vdash s\ A : B[X := A]}\ \forall^2\text{-E}\right)^* \quad \equiv \quad \frac{\overset{\vdots\ \delta^*}{\Gamma^* \vdash \forall X.\, |s| \in B^*}}{\Gamma^* \vdash |s| \in B^*[X := A^*]}\ \forall^2\text{-E}$$

Fig. 3. The Reynolds embedding

Formulas

$$\begin{array}{lcl}(M \in X)^\circ & \equiv & X \\ (\phi \to \psi)^\circ & \equiv & \phi^\circ \to \psi^\circ \\ (\forall x.\, \psi)^\circ & \equiv & \psi^\circ \\ (\forall X.\, \psi)^\circ & \equiv & \forall X.\, \psi^\circ \end{array}$$

Contexts

$$(\phi_1, \ldots, \phi_n)^\circ \;\equiv\; z_1 \in \phi_1^\circ, \ldots, z_n \in \phi_n^\circ$$

Judgements

$$(\Theta \vdash \phi)^\circ \;\equiv\; \Theta^\circ \vdash t : \phi^\circ$$

Proofs

$$\left(\frac{}{\phi_1, \ldots, \phi_n \vdash \phi_i}\,\mathrm{Id}\right)^\circ \quad\equiv\quad \frac{}{z_1 : \phi_1^\circ, \ldots, z_n : \phi_n^\circ \vdash z_i : \phi_i^\circ}\,\mathrm{Id}$$

$$\left(\frac{\begin{array}{c}\vdots\,\pi\\ \Theta,\, \phi \vdash \psi\end{array}}{\Theta \vdash \phi \to \psi}\,{\to}\text{-I}\right)^\circ \quad\equiv\quad \frac{\begin{array}{c}\vdots\,\pi^\circ\\ \Theta^\circ,\, z : \phi^\circ \vdash u : \psi^\circ\end{array}}{\Theta^\circ \vdash \lambda z{:}\phi^\circ.\, u : \phi^\circ \to \psi^\circ}\,{\to}\text{-I}$$

$$\left(\frac{\begin{array}{cc}\vdots\,\pi & \vdots\,\rho\\ \Theta \vdash \phi \to \psi & \Theta \vdash \phi\end{array}}{\Theta \vdash \psi}\,{\to}\text{-E}\right)^\circ \quad\equiv\quad \frac{\begin{array}{cc}\vdots\,\pi^\circ & \vdots\,\rho^\circ\\ \Theta^\circ \vdash s : \phi^\circ \to \psi^\circ & \Theta^\circ \vdash t : \phi^\circ\end{array}}{\Theta^\circ \vdash s\; t : \psi^\circ}\,{\to}\text{-E}$$

$$\left(\frac{\begin{array}{c}\vdots\,\pi\\ \Theta \vdash \psi\end{array}}{\Theta \vdash \forall x.\, \psi}\,\forall^1\text{-I}\right)^\circ \quad\equiv\quad \begin{array}{c}\vdots\,\pi^\circ\\ \Theta^\circ \vdash u : \psi^\circ\end{array}$$

$$\left(\frac{\begin{array}{c}\vdots\,\pi\\ \Theta \vdash \forall x.\, \psi\end{array}}{\Theta \vdash \psi[x := M]}\,\forall^1\text{-E}\right)^\circ \quad\equiv\quad \begin{array}{c}\vdots\,\pi^\circ\\ \Theta^\circ \vdash s : \psi^\circ\end{array}$$

$$\left(\frac{\begin{array}{c}\vdots\,\pi\\ \Theta \vdash \psi\end{array}}{\Theta \vdash \forall X.\, \psi}\,\forall^2\text{-I}\right)^\circ \quad\equiv\quad \frac{\begin{array}{c}\vdots\,\pi^\circ\\ \Theta^\circ \vdash u : \psi^\circ\end{array}}{\Theta^\circ \vdash \Lambda X.\, u : \forall X.\, \psi^\circ}\,\forall^2\text{-I}$$

$$\left(\frac{\begin{array}{c}\vdots\,\pi\\ \Theta \vdash \forall X.\, \psi\end{array}}{\Theta \vdash \psi[X := \{\, x \mid \phi \,\}]}\,\forall^2\text{-E}\right)^\circ \quad\equiv\quad \frac{\begin{array}{c}\vdots\,\pi^\circ\\ \Theta^\circ \vdash s : \forall X.\, \psi^\circ\end{array}}{\Theta^\circ \vdash s\; \phi^\circ : \psi^\circ[X := \phi^\circ]}\,\forall^2\text{-E}$$

$$\left(\frac{\begin{array}{c}\vdots\,\pi\\ \Theta \vdash \psi[x := M]\end{array}}{\Theta \vdash \psi[x := N]}\,\beta\right)^\circ \quad\equiv\quad \begin{array}{c}\vdots\,\pi^\circ\\ \Theta^\circ \vdash \psi^\circ\end{array}$$

Fig. 4. The Girard projection

Formulas

$$
\begin{array}{lcl}
(M \in X)^\ddagger & \equiv & (M, M') \in X^\ddagger \\
(\phi \to \psi)^\ddagger & \equiv & \phi^\ddagger \to \psi^\ddagger \\
(\forall x.\, \psi)^\ddagger & \equiv & \forall x.\, \forall x'.\, \psi^\ddagger \\
(\forall X.\, \psi)^\ddagger & \equiv & \forall X^\ddagger.\, \psi^\ddagger
\end{array}
$$

Contexts

$$
(\phi_1, \ldots, \phi_n)^\ddagger \;\equiv\; \phi_1^\ddagger, \ldots, \phi_n^\ddagger
$$

Judgements

$$
(\Theta \vdash \phi)^\ddagger \;\equiv\; \Theta^\ddagger \vdash \phi^\ddagger
$$

Proofs

$$
\left(\frac{}{\phi_1, \ldots, \phi_n \vdash \phi_i}\,\mathrm{Id}\right)^\ddagger \quad\equiv\quad \frac{}{\phi_1^\ddagger, \ldots, \phi_n^\ddagger \vdash \phi_i^\ddagger}\,\mathrm{Id}
$$

$$
\left(\frac{\begin{array}{c}\vdots\,\pi\\ \Theta,\, \phi \vdash \psi\end{array}}{\Theta \vdash \phi \to \psi}\,{\to}\text{-I}\right)^\ddagger \quad\equiv\quad \frac{\begin{array}{c}\vdots\,\pi^\ddagger\\ \Theta^\ddagger,\, \phi^\ddagger \vdash \psi^\ddagger\end{array}}{\Theta^\ddagger \vdash \phi^\ddagger \to \psi^\ddagger}\,{\to}\text{-I}
$$

$$
\left(\frac{\begin{array}{cc}\vdots\,\pi & \vdots\,\rho\\ \Theta \vdash \phi \to \psi & \Theta \vdash \phi\end{array}}{\Theta \vdash \psi}\,{\to}\text{-E}\right)^\ddagger \quad\equiv\quad \frac{\begin{array}{cc}\vdots\,\pi^\ddagger & \vdots\,\rho^\ddagger\\ \Theta^\ddagger \vdash \phi^\ddagger \to \psi^\ddagger & \Theta^\ddagger \vdash \phi^\ddagger\end{array}}{\Theta^\ddagger \vdash \psi^\ddagger}\,{\to}\text{-E}
$$

$$
\left(\frac{\begin{array}{c}\vdots\,\pi\\ \Theta \vdash \psi\end{array}}{\Theta \vdash \forall x.\, \psi}\,\forall^1\text{-I}\right)^\ddagger \quad\equiv\quad \frac{\begin{array}{c}\vdots\,\pi^\ddagger\\ \Theta^\ddagger \vdash \psi^\ddagger\end{array}}{\Theta^\ddagger \vdash \forall x.\, \forall x'.\, \psi^\ddagger}\,\forall^1\text{-I (twice)}
$$

$$
\left(\frac{\begin{array}{c}\vdots\,\pi\\ \Theta \vdash \forall x.\, \psi\end{array}}{\Theta \vdash \psi[x := M]}\,\forall^1\text{-E}\right)^\ddagger \quad\equiv\quad \frac{\begin{array}{c}\vdots\,\pi^\ddagger\\ \Theta^\ddagger \vdash \forall x.\, \forall x'.\, \psi^\ddagger\end{array}}{\Theta^\ddagger \vdash \psi^\ddagger[x := M, x' := M']}\,\forall^1\text{-E (twice)}
$$

$$
\left(\frac{\begin{array}{c}\vdots\,\pi\\ \Theta \vdash \psi\end{array}}{\Theta \vdash \forall X.\, \psi}\,\forall^2\text{-I}\right)^\ddagger \quad\equiv\quad \frac{\begin{array}{c}\vdots\,\pi^\ddagger\\ \Theta^\ddagger \vdash \psi^\ddagger\end{array}}{\Theta^\ddagger \vdash \forall X^\ddagger.\, \psi^\ddagger}\,\forall^2\text{-I}
$$

$$
\left(\frac{\begin{array}{c}\vdots\,\pi\\ \Theta \vdash \forall X.\, \psi\end{array}}{\Theta \vdash \psi[X := \{\, x \mid \phi \,\}]}\,\forall^2\text{-E}\right)^\ddagger \quad\equiv\quad \frac{\begin{array}{c}\vdots\,\pi^\ddagger\\ \Theta^\ddagger \vdash \forall X^\ddagger.\, \psi^\ddagger\end{array}}{\Theta^\ddagger \vdash \psi^\ddagger[X^\ddagger := \{\, (x, x') \mid \phi^\ddagger \,\}]}\,\forall^2\text{-E}
$$

$$
\left(\frac{\begin{array}{c}\vdots\,\pi\\ \Theta \vdash \psi[x := M]\end{array}}{\Theta \vdash \psi[x := N]}\,\beta\right)^\ddagger \quad\equiv\quad \frac{\begin{array}{c}\vdots\,\pi^\ddagger\\ \Theta^\ddagger \vdash \psi^\ddagger[x := M,\, x' := M']\end{array}}{\Theta^\ddagger \vdash \psi^\ddagger[x := N,\, x' := N']}\,\beta\text{ (twice)}
$$

Fig. 5. The doubling embedding

Syntax

$$\begin{array}{lll} \text{Type variables} & X, Y, Z & \\ \text{Algebraic types} & T, U & ::= Z \mid \mathrm{Nat} \mid 0 \mid T + U \mid 1 \mid T \times U \mid \mu Z.\, T \end{array}$$

Inductive interpretation

$$\begin{array}{lcl} \lceil \mathrm{Nat} \rceil & \equiv & \{\, n \mid \forall Z.\, (\forall m.\, m \in Z \to \mathsf{succ}\ m \in Z) \to \mathsf{zero} \in Z \to n \in Z \,\} \\ \lceil 0 \rceil & \equiv & \{\, z \mid \forall Z.\, z \in Z \,\} \\ \lceil T + U \rceil & \equiv & \{\, z \mid \forall Z.\, (\forall x.\, x \in \lceil T \rceil \to \mathsf{inl}\ x \in Z) \to (\forall y.\, y \in \lceil U \rceil \to \mathsf{inr}\ y \in Z) \to z \in Z \,\} \\ \lceil 1 \rceil & \equiv & \{\, z \mid \forall Z.\, \mathsf{unit} \in Z \to z \in Z \,\} \\ \lceil T \times U \rceil & \equiv & \{\, z \mid \forall Z.\, (\forall x.\, x \in \lceil T \rceil \to \forall y.\, y \in \lceil U \rceil \to \mathsf{pair}\ x\ y \in Z) \to z \in Z \,\} \\ \lceil \mu Z.\, T \rceil & \equiv & \{\, z \mid \forall Z.\, (\forall x.\, x \in \lceil T[Z := \mu Z.\, T] \rceil \to \mathsf{in}\ x \in Z) \to z \in Z \,\} \end{array}$$

Deductive interpretation

$$\begin{array}{lcl} \lfloor \mathrm{Nat} \rfloor & \equiv & \forall Z.\, (Z \to Z) \to Z \to Z \\ \lfloor 0 \rfloor & \equiv & \forall Z.\, Z \\ \lfloor T + U \rfloor & \equiv & \forall Z.\, (T \to Z) \to (U \to Z) \to Z \\ \lfloor 1 \rfloor & \equiv & \forall Z.\, Z \to Z \\ \lfloor T \times U \rfloor & \equiv & \forall Z.\, (T \to U \to Z) \to Z \\ \lfloor \mu Z.\, T \rfloor & \equiv & \forall Z.\, (T \to Z) \to Z \end{array}$$

Constructors

$$\begin{array}{lcl} \mathbf{zero} & : & \lfloor \mathrm{Nat} \rfloor \\ \mathbf{zero} & = & \Lambda Z.\, \lambda s : Z \to Z.\, \lambda z : Z.\, z \\ \\ \mathbf{succ} & : & \lfloor \mathrm{Nat} \rfloor \to \lfloor \mathrm{Nat} \rfloor \\ \mathbf{succ} & = & \lambda n : \lfloor \mathrm{Nat} \rfloor\, .\, \Lambda Z.\, \lambda s : Z \to Z.\, \lambda z : Z.\, s\ (n\ Z\ s\ z) \\ \\ \mathbf{inl} & : & \lfloor T \rfloor \to \lfloor T + U \rfloor \\ \mathbf{inl} & = & \lambda x : \lfloor T \rfloor\, .\, \Lambda Z.\, \lambda k : \lfloor T \rfloor \to Z.\, \lambda l : \lfloor U \rfloor \to Z.\, k\ x \\ \\ \mathbf{inr} & : & \lfloor U \rfloor \to \lfloor T + U \rfloor \\ \mathbf{inr} & = & \lambda y : \lfloor U \rfloor\, .\, \Lambda Z.\, \lambda k : \lfloor T \rfloor \to Z.\, \lambda l : \lfloor U \rfloor \to Z.\, l\ y \\ \\ \mathbf{unit} & : & \lfloor 1 \rfloor \\ \mathbf{unit} & = & \Lambda Z.\, \lambda z : Z.\, z \\ \\ \mathbf{pair} & : & \lfloor T \rfloor \to \lfloor U \rfloor \to \lfloor T \times U \rfloor \\ \mathbf{pair} & = & \lambda x : \lfloor T \rfloor\, .\, \lambda y : \lfloor U \rfloor\, .\, \Lambda Z.\, \lambda k : \lfloor T \rfloor \to \lfloor U \rfloor \to Z.\, k\ x\ y \\ \\ \mathbf{in} & : & \lfloor T[Z := \mu Z.\, T] \rfloor \to \lfloor \mu Z.\, T \rfloor \\ \mathbf{in} & = & \lambda x : \lfloor T[Z := \mu Z.\, T] \rfloor\, .\, \lambda Z.\, \lambda k : \lfloor T \rfloor \to Z.\, k\ (\lfloor T \rfloor\ [\lambda x : \lfloor \mu Z.\, T \rfloor\, .\, x\ Z\ k]\ x) \end{array}$$

Fig. 6. Algebraic types