Logical Relations and Data Abstraction*

John Power**[1] and Edmund Robinson***[2]

[1] Laboratory for the Foundations of Computer Science, University of Edinburgh
ajp@dcs.ed.ac.uk
[2] Department of Computer Science, Queen Mary and Westfield College,
edmundr@dcs.qmw.ac.uk

Abstract. We prove, in the context of simple type theory, that logical relations are sound and complete for data abstraction as given by equational specifications. Specifically, we show that two implementations of an equationally specified abstract type are equivalent if and only if they are linked by a suitable logical relation. This allows us to introduce new types and operations of any order on those types, and to impose equations between terms of any order. Implementations are required to respect these equations up to a general form of contextual equivalence, and two implementations are equivalent if they produce the same contextual equivalence on terms of the enlarged language. Logical relations are introduced abstractly, soundness is almost automatic, but completeness is more difficult, achieved using a variant of Jung and Tiuryn's logical relations of varying arity. The results are expressed and proved categorically.

1 Introduction

Logical relations are a standard tool for establishing the equivalence of data representations. If one can find a family of relations on corresponding types which is a congruence, i.e., are preserved by operations of the theory, and reduces to equality on observable types, then clearly observable operations will be equal in the two representations. Such a family need not form a logical relation (it may form a lax or pre-logical relation [4, 12]), but logical relations do provide a way to construct such families by induction on type structure. So the establishment of a logical relation suffices to determine equivalence of representation at least for simple types. It has long been known that for algebraic theories with first order operations and equations, equivalent representations are indeed linked by a logical relation, see Mitchell [9, 10]. It has also long been known that standard logical relations are not complete in this sense for higher-order theories.

* **Keywords:** logical relations, cartesian closed fibrations, interpretations, lambda calculus.

** This work has been done with the support of EPSRC grant GR/M56333 and a British Council grant.

*** This author would like to acknowledge the support of an EPSRC Advanced Fellowship, EPSRC grant GR/L54639, and ESPRIT Working Group 6811 Applied Semantics

The purpose of this paper is to show that by relaxing the definition of logical relation, we can extend Mitchell's completeness result to higher-order theories.

The approach we take here has two major sources. The first is the work of Achim Jung and Jerzy Tiuryn [5], who used a form of logical relations to characterise lambda definability in the simple type hierarchy. The second is the thesis of Claudio Hermida [3], which explores the connection between logical predicates and logical formalisms expressed as fibrations defined over the semantic category. Sadly there is still no really accessible reference for the material and ideas in this thesis.

From a practical angle, the notion of abstract type we use here is quite restricted as we only consider equationally specified types, albeit with equations between higher-order operators. This is also the level of generality of Alimohamed's work on definability [1]. However, it raises a natural question of whether the link with logic exploited in this paper could be further exploited to allow a less restrictive specification language.

Finally, if data abstraction is the study of equivalence of representation, then data refinement is the study of improvements. The two ideas are clearly linked, and the fundamental notion of this paper, that of being "linked by a logical relation", is closely related to the notion of L-relation in [6] (see also [14]). The work of that paper, which is also about data equivalence but from a purely categorical perspective, and the work of this paper, and those on lax and prelogical relations [4, 12] clearly need to be combined at some point.

A natural question, however, is why we chose to base our work on cartesian closed categories rather than Henkin models.

Henkin models provide a natural extension of classical set-based first-order model theory to the higher-order setting of simply typed lambda calculi. Much of their appeal lies in the immediacy of this relation. Cartesian closed categories, on the other hand, arose out of the desire to express in purely category theoretic terms the characteristic properties of function spaces such as those found in Set. The formal link with typed lambda calculus, though not surprising in itself, came surprisingly late (cf. Lambek and Scott [7]), and prompted a slight redefinition of the notion to allow categories which had products but not arbitrary finite limits.

Cartesian closed categories and Henkin models are intimately linked. If one starts with a Henkin model of $\lambda{\rightarrow}$ one can easily extend it to model the type theory with products, $\lambda{\times}{\rightarrow}$ (using the fact that the type $(A_1 \times \ldots \times A_n) \rightarrow (B_1 \times \ldots \times B_m)$ is isomorphic to a product of pre-existing types $(A_1 \rightarrow (\ldots (A_n \rightarrow B_1)\ldots)) \times \ldots \times (A_1 \rightarrow (\ldots (A_n \rightarrow B_m)\ldots)))$. If one has a Henkin model of typed lambda calculus with products, then one can obtain a (concrete) cartesian closed category. The objects are types and the morphisms $A \longrightarrow B$ are the elements of the function type $[A \rightarrow B]$ in the Henkin model. Conversely if one has a cartesian closed category C, together with a product-preserving functor $U : C \longrightarrow \mathsf{Set}$, then one obtains an applicative structure with product types. The types are the objects of the category, and the type A is interpreted as the set UA. This automatically has representations of the combinators because of their presence as morphisms in the cartesian closed category. However it is not necessarily a Henkin model because in general different elements of the function type may represent the same function. If, however, C is a concrete category and U its underlying set functor, this will not happen. In that case we do get a Henkin model.

So, from this perspective, cartesian closed categories generalise Henkin models. The question is whether that generality, or that difference in structure, is useful. We believe that it is.

One technical difference between the two approaches is apparent if one thinks of semantics taken in some kind of sets with structure (e.g. domains). It is natural to ask when this collection of sets forms a model of the simply typed lambda calculus. The category-theoretic answer is that one has to specify an appropriate notion of homomorphism. The resulting category is then either cartesian closed, in which case it supports a model structure, or it is not, when it doesn't. Moreover, if it is cartesian closed then the structure we use to model the lambda calculus is essentially unique. In contrast, it is much less clear what the constraints on a Henkin structure might be, or in what sense the structure would be unique if it existed. Essentially the development has to mimic the category-theoretic. First one defines an appropriate notion of homomorphism. Then one asks whether the set of homomorphisms between two objects can be made to carry the structure of the objects we are interested in. Or rather, since it almost always can, whether it can be made to do so in a way that makes application and the combinators homomorphisms. It is perhaps because of this apparently greater complexity that domain theorists and others talk about specific categories of domains being cartesian closed, and not about Henkin structures on classes of domains.

Moreover, Henkin models can only talk about models built on sets. This constraint is not always natural. For example it is natural to say that the canonical way of viewing a partially-ordered set as a category turns a Heyting (or Boolean) algebra into a cartesian closed category. It is not natural to say that they carry a Henkin model structure, since that requires presenting the elements of the partial order as themselves sets. A further example comes from the Curry-Howard correspondence. This can be conveniently and precisely expressed by saying that there is an isomorphism between a category whose objects are types and whose morphisms are $\beta\eta$-equivalence classes of λ-terms, and a category whose objects are propositions and whose morphisms are normalisation classes of proofs. The Henkin model account has to be forced to apply to this, most notably because in the event that A is not provable, then a proof of $A \Rightarrow B$ cannot adequately be described as a function which takes proofs of A to proofs of B.

We now turn to possible directions for our own future work. First, there are links between this work and the work of Honsell and Sannella on pre-logical relations. Unfortunately a preliminary account of these links had to be cut from this paper for reasons of space. We intend to return to this theme. An obvious direction in which to extend this work is to address models of call-by-value lambda-calculi such as the computational lambda-calculus [11].

Several people have been of considerable help in making comments or suggestions: they include Adam Eppendahl, Claudio Hermida, Peter O'Hearn, David Pym, Makoto Takeyama, and Bob Tennent. We would also like to thank the (many) anonymous referees of the original paper. We are sorry that we have not been able to make all the improvements they suggested.

2 Contextual equivalence

In this section, we give a categorical account of a standard Morris-style contextual equivalence. As we are working in the context of simple type theory, and the theory may well be strongly normalising, we use a denotational rather than an operational approach, and observe values produced by computations rather than termination.

The basic structure is as follows. We assume a given cartesian closed category C. For instance, C may be the category Set of small sets, or, if we want to be sure to include fixed point operators to model recursion, we could consider C to be the category of ω-cpo's with least element, and with maps being functions that preserve the poset structure and sup's of ω-chains. Other alternatives for C include categories generated by axiomatic domain theory [2], or presheaf categories, or toposes. We simply require that C be cartesian closed. The category C will represent our base category in which we take our denotational semantics.

We further assume that we are given a set of observable pairs of types $Obs \subseteq ob(C) \times ob(C)$. The intuition behind this is that we are allowed to form judgements of equality

$$\boldsymbol{x} : A \vdash e = e' : B$$

where $(A, B) \epsilon Obs$. An alternative approach is to use a set of observable types, but we think this accords better with our open-term interpretation of the lambda calculus, and will allow a simpler extension to dependent type theory.

We use the structure of observables to generate a contextual equivalence $\approx$ on C, which allows us to compare $f \approx g$ when f and g are maps with common domain and codomain: type-theoretically, this amounts to demanding that contextually equivalent terms have the same type and are defined in the same context.

Example 1.
1. The underlying category is CPO, and $Obs = \{(1, 1_\perp)\}$. This gives one of the standard forms of Morris-style contextual equivalence, in which we are allowed to observe termination of closed terms.
2. The underlying category is Set, and $Obs = \{(1, Bool)\}$. Here, there is no non-termination, but contexts produce closed terms of type $Bool$. We can observe the boolean produced.
3. $Obs = ob(C) \times ob(C)$. In this case everything is observable. Not surprisingly $\approx$ is just equality.

Definition 1. *Given $f, g : X \longrightarrow Y$, we define $\sim$ by $f \sim g$ if for all $(A, B)\epsilon Obs$, and for all $\alpha : A \longrightarrow X$ and $\beta : Y \longrightarrow B$, we have*

$$A \xrightarrow{\alpha} X \xrightarrow{f} Y \xrightarrow{\beta} B$$

equals

$$A \xrightarrow{\alpha} X \xrightarrow{g} Y \xrightarrow{\beta} B$$

The category theoretic definition of a congruence is often used to mean an equivalence relation on each homset of a category C such that that equivalence relation is respected by composition on either side. Here, by a congruence, we mean that notion together with invariance under products and exponentials. That allows us to make the primary definition of this section as follows.

Definition 2. *We define $\approx$ to be the largest congruence contained in $\sim$.*

Mercifully, we can characterise $\approx$ more directly by means of the following proposition. We denote the exponential of X raised to the power U by $[U, X]$. We follow this convention for exponentials in a category, for exponentials of not only objects but maps, and for exponentials of categories, i.e., for functor categories, when we need them.

Proposition 1. *Given $f, g : X \longrightarrow Y$, the following are equivalent:*

1. $f \approx g$
2. *for all U, V in $ob(C)$, we have $[U, f] \times V \sim [U, g] \times V$*

To prove this proposition, apply the following lemma.

Lemma 1. *Let $\sim$ be an equivalence relation on parallel pairs of arrows of a category C. Then*

1. *composition preserves $\sim$, and hence $C/\sim$ is a category and $C \longrightarrow C/\sim$ is a functor if and only $f \sim g$ implies $\beta \cdot f \cdot \alpha \sim \beta \cdot g \cdot \alpha$.*
2. *if C has finite products, then $C/\sim$ has finite products, and passage to the quotient preserves them if and only if 1 holds and $f \sim g$ implies $f \times V \sim g \times V$ for all V in $ob(C)$.*
3. *if C is cartesian closed, then $C/\sim$ is a cartesian closed category and passage to the quotient is a cartesian closed functor if and only if 1 and 2 hold and $f \sim g$ implies $[U, f] \sim [U, g]$ for all U in $ob(C)$, or equivalently if and only if 1 holds and $f \sim g$ implies $[U, f] \times V \sim [U, g] \times V$ for all U,V in $ob(C)$.*

We can also make precise the relationship with Morris-style contextual equivalence. Any cartesian closed category can be regarded as a model of an applied lambda calculus in which we have base types corresponding to the objects of the category, and for each morphism $f : X \longrightarrow Y$, we are given a term constructor (function) $\hat{f}$, such that if e is a term of type X, then $f(e)$ is a term of type Y (as in Lambek and Scott [7]). As usual, a context, $\mathcal{C}[-]$ is a term with a hole (written $[-]$), such that when we plug a term of suitable type into the hole, we get something that we can run and observe. Making this precise for typed open terms is always messy. The simplest solution is to consider terms in context. In that case, we demand

$$\frac{x : X \vdash e : Y}{a : A \vdash \mathcal{C}[e] : B}$$

should be an admissible rule for the type system (here, in order to make $\mathcal{C}[e]$ directly observable, $(A, B) \in Obs$). This can be achieved by ensuring that

$$\frac{x : X \vdash [-] : Y}{a : A \vdash \mathcal{C}[-] : B}$$

is a derived rule.

Definition 3. *Two terms e and e' (more accurately, two terms in the same context: $x : X \vdash e : Y$ and $x : X \vdash e' : Y$ ...) are contextually equivalent, $e \cong e'$ if and only if for all contexts $\mathcal{C}[-]$, $M\{\mathcal{C}[e]\} = M\{\mathcal{C}[e']\}$, where $M\{e\}$ is the semantics of e as taken in the category C.*

Proposition 2. $e \cong e'$ *if and only if* $M\{e\} \approx M\{e'\}$.

Proof. In the if direction, this is a simple structural induction on the context. In the other, it depends on the expressibility of substitution via abstraction, application, and beta reduction, and on the fact that all morphisms in the category are expressible as terms in the calculus.

3 Abstract types

An equationally specified abstract type consists of a new type equipped with operations linking it to pre-existing types, and certain equations on terms built from those operations. The canonical example is `stack`. For our purposes, we may as well have a number of new types, and we interpret the operations as (open) terms in context

$$\boldsymbol{x} : \boldsymbol{X} \vdash f : Y$$

where the $\boldsymbol{X}$ and Y may be arbitrary type expressions built up from old and new types. This generates an extended lambda calculus in which the equations are interpreted as equality judgements

$$\boldsymbol{x} : \boldsymbol{X} \vdash e = e' : Y$$

Such a system freely generates a cartesian closed category E which comes with a cartesian closed functor $i : C \longrightarrow E$. This functor is injective on objects, but in general is neither full nor faithful as we can add extra operations between pre-existing types and impose new equations between pre-existing operations. Conversely, given any injective on objects cartesian closed functor $i : C \longrightarrow E$, the category E is equivalent to a category constructed from C in this fashion, as we may generate it using all objects in $E \backslash i(C)$ and take all valid equations. So we can take $i : C \longrightarrow E$ as the extension of C by some abstract types.

We split this construction into two stages: first adjoin types and operations freely to give $i : C \longrightarrow D$, and then quotient by the equations to give $q : D \longrightarrow D/\approx_E$. In fact, we shall never use the free property of D and can take it to be an arbitrary cartesian closed category. The free property of such a D is fundamental to the work generalising logical relations for an account of data refinement in [6].

An implementation of the abstract type back in C is given by a cartesian closed functor $F : D \longrightarrow C$. We do not want to change the interpretation of the operations in C, so require $F \cdot i = id_C$. We also want the interpretation to validate the equations of the abstract type, at least up to a suitable notion of observational congruence. We illustrate the problems with the canonical simple (but first-order) example:

Example 2. Let C be Set and let D consist of Set together with the formal addition of a type stack together with its operations (empty, top, pop and push) and equations ($\mathsf{pop}(\mathsf{push}(n, s)) = s$, and $\mathsf{top}(\mathsf{push}(n, s)) = n$). An implementation amounts to a realisation of the type stack together with its operations and subject to its equations in Set in both cases. Consider an array implementation, in which stack is implemented by an (unbounded) array and a top pointer, and pop simply decrements the top pointer, leaving the array unchanged. In this implementation, the interpretation of $\mathsf{pop}(\mathsf{push}(n, s))$ is not equal to s. Moreover, in Set we can apply an operation which reads the stack one place above the top pointer, and hence observe the difference. Of course this violates the stack abstraction.

The example above tells us that we cannot make the obvious restriction on F: if $e = e'$ is an equation of the abstract type, then $F(M\{e\}) \approx F(M\{e'\})$. Instead we must build in the notion that contexts have restricted access to abstract types. This leads to the following definition (cf. definitions 1 and 2):

Definition 4.
1. *Suppose F is a functor $D \longrightarrow C$, and $f, g : X \longrightarrow Y$ is a parallel pair of morphisms in D, then $f \sim_F g$ if and only if for all (A, B) in Obs, and for all $\alpha : iA \longrightarrow X$ and $\beta : Y \longrightarrow iB$, we have $F(\beta \cdot f \cdot \alpha) = F(\beta \cdot g \cdot \alpha)$.*
2. *If, in addition, C, D and F are cartesian closed, then $\approx_F$ is the largest congruence on D contained in $\sim_F$ (i.e. $f \approx_F g$ if and only if for all U,V in $ob(D)$, $([U, f] \times V) \sim_f ([U, g] \times V)$).*

The implementation is valid if and only if all equations $\boldsymbol{x} : \boldsymbol{X} \vdash e = e' : Y$ hold up to contextual equivalence, i.e., $M\{e\} \approx_F M\{e'\}$, or equivalently, the quotient $D \longrightarrow D/\approx_F$ factors through $D \longrightarrow D/\approx_E$. The only function of the equational theory is to restrict the notion of implementation. It has no effect on when two implementations are equivalent, and so now plays no further role in the story: it just, so to speak, comes along for the ride.

Definition 5. *Suppose $i : C \longrightarrow D$ is an injective-on-objects cartesian closed functor between cartesian-closed categories, and $\mathcal{E}$ is a set of parallel pairs of morphisms in D (which we view as the equations of an abstract type system), then an implementation F of $(D, \mathcal{E})$ in C is a cartesian closed functor $F : D \longrightarrow C$, such that $F \cdot i = id_C$, and for all $(f, g) \in \mathcal{E}$, $F(f) \approx_F F(g)$.*

We now observe some properties of $\approx_F$ that will help us to make precise what we mean by two implementations being equivalent.

Proposition 3. *Let $F : D \longrightarrow C$ be a cartesian closed functor such that $F \cdot i = id_C$. Then the following hold of $\approx_F$:*

1. *for all $f, g : X \longrightarrow Y$ in D, if $Ff \approx Fg$, then $f \approx_F g$.*
2. *for all $f, g : X \longrightarrow Y$ in C, if $f \approx g$, then $if \approx_F ig$.*
3. *for all $f, g : iX \longrightarrow iY$ in D, we have $f \approx_F g$ if and only if $Ff \approx Fg$.*
4. *for all $f, g : iA \longrightarrow iB$ in D, where $(A, B) \epsilon Obs$, we have $f \approx_F g$ if and only if $Ff = Fg$.*

Proof. 1. Suppose $Ff \approx Fg$. Then, given $\alpha : iA \longrightarrow [U, X] \times V$ and $\beta : [U, Y] \times V \longrightarrow iB$, we have $F(\beta \cdot ([U, f] \times V) \cdot \alpha) = F\beta \cdot ([FU, Ff] \times FV) \cdot F\alpha = F\beta \cdot ([FU, Fg] \times FV) \cdot F\alpha = F(\beta \cdot ([U, g] \times V) \cdot \alpha)$
2. Suppose $f \approx g$. Then, given $\alpha : iA \longrightarrow [U, iX] \times V$ and $\beta : [U, iY] \times V \longrightarrow iB$, we have $F(\beta \cdot ([U, if] \times V) \cdot \alpha) = F\beta \cdot ([FU, Fif] \times FV) \cdot F\alpha = F\beta \cdot ([FU, f] \times FV) \cdot F\alpha = F\beta \cdot ([FU, g] \times FV) \cdot F\alpha = F\beta \cdot ([FU, Fig] \times FV) \cdot F\alpha = F(\beta \cdot ([U, ig] \times V) \cdot \alpha)$
3. Suppose $f \approx_F g$. Then, given $\alpha : A \longrightarrow [U, X] \times V$ and $\beta : [U, Y] \times V \longrightarrow B$, we have $\beta \cdot ([U, Ff] \times V) \cdot \alpha = Fi\beta \cdot ([FiU, Ff] \times FiV) \cdot Fi\alpha = F(i\beta \cdot ([iU, f] \times iV) \cdot i\alpha) = F(i\beta \cdot ([iU, g] \times iV) \cdot i\alpha) = Fi\beta \cdot ([FiU, Fg] \times FiV) \cdot Fi\alpha = \beta \cdot ([U, Fg] \times V) \cdot \alpha$. The converse has already been shown.
4. $Ff \approx Fg$ if and only if $Ff = Fg$ for observable types.

Armed with these observations about the interaction of the definitions of $\approx$ and $\approx_F$, we can now establish relationships between a pair of interpretations F and G, where we understand an interpretation to be defined to be a functor $F : D \longrightarrow C$ such that $F \cdot i = id_C$.

Proposition 4. *Let $F, G : D \longrightarrow C$ be two interpretations. Then, the following are equivalent:*

1. *F and G induce the same equivalence: $\approx_F = \approx_G$*
2. *F and G induce the same equivalence on observable operations: $\approx_F |_{iObs} = \approx_G |_{iObs}$.*
3. *the interpretations of operations between pre-existing types are contextually equivalent: for all $f : iX \longrightarrow iY$, we have $Ff \approx Gf$.*
4. *the interpretations of observable operations are contextually equivalent (in fact, equal): for all $f : iA \longrightarrow iB$ where $(A, B) \epsilon Obs$, we have $Ff \approx Gf$.*

Proof. $1 \Rightarrow 2$ and $3 \Rightarrow 4$ are immediate.

$2 \Rightarrow 1$. Suppose $f \approx_F g : X \longrightarrow Y$, and we have $\alpha : iA \longrightarrow [U, X] \times V$ and $\beta : [U, Y] \times V \longrightarrow iB$. Then $F(\beta \cdot ([U, f] \times V) \cdot \alpha) = F(\beta \cdot ([U, g] \times V) \cdot \alpha)$. But these are between observable types, hence by Proposition 3(4), we have $\beta \cdot ([U, f] \times V) \cdot \alpha \approx_F \beta \cdot ([U, g] \times V) \cdot \alpha$. So $\beta \cdot ([U, f] \times V) \cdot \alpha \approx_G \beta \cdot ([U, g] \times V) \cdot \alpha$; and hence by Proposition 3(4) again, we have $G(\beta \cdot ([U, f] \times V) \cdot \alpha) = G(\beta \cdot ([U, g] \times V) \cdot \alpha)$.

$1 \Rightarrow 3$. Given $f : iX \longrightarrow iY$, we have $Ff = FiFf$, so by Proposition 3(1), we have $f \approx_F iFf$. So $f \approx_G iFf$, and hence by Proposition 3(3), we have $Gf \approx GiFf = Ff$.

$4 \Rightarrow 2$. Given $f, g : iA \longrightarrow iB$ where $(A, B) \epsilon Obs$, suppose $f \approx_F g$. Then $Ff = Fg$. But by hypothesis, $Ff = Gf$, and $Fg = Gg$, so $Gf = Gg$. Hence $f \approx_G g$.

Any of the above conditions would be a reasonable definition of equivalence of implementation. So the proposition conveniently justifies the following.

Definition 6. *Two interpretations $F, G : D \longrightarrow C$ are* equivalent *if they satisfy the equivalent conditions of Proposition 4.*

Example 3.
1. Suppose F and G correspond to two implementations of stack, in both of which the underlying type of stack is the product of an unbounded array of integers (giving the stack), and a non-negative integer (giving the top pointer). In F, pop is implemented by decrementing the top pointer, and leaving the stack unchanged. In G, the top pointer is decremented, and the cell popped is set to zero in the array. F and G are otherwise identical. Then F and G are equivalent implementations.
2. It is not necessary for the underlying types to be the same in the two implementations, if F is the array implementation of stack mentioned above, and G is a list implementation, then F and G will be equivalent exactly when they agree about top(empty) and pop(empty).

4 Logical relations

In [8], Ma and Reynolds began by analysing the classical theory of logical relations for simple types. They focussed on the structure needed to prove the fundamental theorem. For binary relations, this amounts to a cartesian closed category B equipped with a cartesian closed functor $p = (p_0, p_1) : B \longrightarrow C \times C$, and that cartesian closed functor is typically a fibration [3]. In their theory, B is constructed concretely as a category of relations, and so comes with a standard diagonal $\Delta : C \longrightarrow B$. This is also a cartesian closed functor.

For instance, consider C being the category Set. Then an object of B consists of a pair (X, Y) of sets together with a subset R of $X \times Y$. The diagonal $\Delta : C \longrightarrow B$ is indeed the diagonal in the usual sense, taking a set X to the subset $\{(x, x) | x \in X\}$ of $X \times X$.

The significance of the fact that B has and all the functors preserve cartesian closed structure, together with the fact that $p = (p_0, p_1) : B \longrightarrow C \times C$ is a fibration, is that logical relations can be defined on base types, and then extended inductively on the syntax of type expressions as usual. See [3] for a careful analysis of this. Technically, we do not need this much structure for soundness, but having it gives a stronger completeness result and a better link in to logics for reasoning about the type systems. So we take this structure as a characterisation of logical relations, but since our categories are more abstract than those of Ma and Reynolds, we require an abstract characterisation of the diagonal. Moreover, we would like the diagonal to characterise contextual equivalence, not equality.

In the standard relational setting, when $R \subseteq X_0 \times X_1$ and $S \subseteq Y_0 \times Y_1$, a map in B from R to S is exactly a pair of maps $f_0 : X_0 \longrightarrow Y_0$ and $f_1 : X_1 \longrightarrow Y_1$, such that $f_0 \times f_1$ maps R into S, i.e. a pair of maps which respects the relations R and S. In our more abstract setting it is not necessarily true that a map in B is given by a pair of maps in C, and so we shall regard a pair of maps (f_0, f_1)

as respecting the “relations” R and S exactly when there is a map $f : R \longrightarrow S$ such that $p_0(f) = f_0$ and $p_1(f) = f_1$. This enables us to pick out the essential property we need of the diagonal functor: that if two maps preserve diagonals, then they are contextually equivalent.

Definition 7. *Let $p = (p_0, p_1) : B \longrightarrow C \times C$ be a cartesian closed functor, and let $\mathcal{D} : C \longrightarrow B$ be a cartesian closed functor such that $p \cdot \mathcal{D} = \Delta : C \longrightarrow C \times C$. Then we say $\mathcal{D}$ is* diagonal *if, for all $f : \mathcal{D}X \longrightarrow \mathcal{D}Y$, it follows that $p_0 f \approx p_1 f$.*

For a formal definition and exposition of the notion of fibration, one may see Claudio Hermida’s thesis [3], half of which is about the relationship between logical relations and fibrations. The idea is that a fibration with structure, in particular cartesian closed structure, provides a category theoretic equivalent to having a predicate logic with which one can build logical relations. Formally, the definition is as follows.

Definition 8. *A functor $p : \mathcal{P} \longrightarrow \mathcal{C}$ is called a* fibration *if for every object X of $\mathcal{P}$ and every map of the form $g : A \longrightarrow pX$ in $\mathcal{C}$, there is an arrow $g^*_X : g^*(X) \longrightarrow X$ in $\mathcal{P}$ such that $p(g^*(X)) = A$ and $p(g^{*X}) = g$, and the following universal property is satisfied:*

for every arrow $f : Z \longrightarrow X$ and every arrow $h : pZ \longrightarrow p(g^*(X))$ such that $g \cdot h = p(f)$, there exists a unique arrow $k : Z \longrightarrow g^*(X)$ such that $p(k) = h$ and $g^*_X \cdot k = f$.

Example 4. We recast our major example of binary relations as follows. $\mathcal{C}$ is $\mathsf{Set} \times \mathsf{Set}$. The objects of $\mathcal{P}$ are binary predicates (formally, a pair of sets (X_0, X_1), together with a predicate $R(x_0, x_1)$ on them). A morphism from $(X_0, X_1, R(x_0, x_1))$ to $(Y_0, Y_1, S(y_0, y_1))$ is a pair of functions (f_0, f_1), such that

$$\models \forall x_0, x_1.\ R(x_0, x_1) \Rightarrow S(f_0(x_0), f_1(x_1))$$

If $X = (X_0, X_1, R(x_0, x_1))$, and $g = (g_0 : Z_0 \longrightarrow X_0, g_1 : Z_1 \longrightarrow X_1)$, then $g^*(X) = (Z_0, Z_1, R(g_0(z_0), g_1(z_1)))$.

Definition 9. *F and G are* linked by a logical relation *if there is a cartesian closed category B and a cartesian closed fibration $p : B \longrightarrow C \times C$ together with a cartesian closed functor $L : D \longrightarrow B$ lying over (F, G) (i.e. $p_0 \cdot L = F$ and $p_1 \cdot L = G$) such that $\mathcal{D} = L \cdot i$ is diagonal.*

$$\begin{tikzcd}
 & & B \arrow[d, "{p = (p_0, p_1)}"] \\
C \arrow[r, "i"'] & D \arrow[ur, "L"] \arrow[r, "{(F,G)}"'] & C \times C
\end{tikzcd}$$

We can now immediately prove the soundness of this form of logical relation for proving equivalence of implementations.

Proposition 5. *If F and G are two implementations which are linked by a logical relation, L, then F and G are equivalent.*

Completeness involves a variant of Jung and Tiuryn's Kripke logical relations of varying arity.

5 Jung-Tiuryn's logical relations of varying arity

In this section, we outline how we prove the completeness part of our main result. Given equivalent interpretations $F, G : D \longrightarrow C$, we seek a cartesian closed fibration $p : B \longrightarrow C \times C$ and a cartesian closed functor from D to B. Our construction is motivated by Jung and Tiuryn's construction of Kripke logical relations of varying arity [5], though it corresponds to a variant and not their precise construction. (It is possible to prove the result we want in a way that corresponds precisely to theirs. The construction is, however, less compact and less obviously generalisable.)

First, Jung and Tiuryn's choice of name is questionable. Their paper is concerned with definability in the simple type hierarchy. Definability is a unary predicate, and there is a precise sense in which the relations they define are, despite appearances to the contrary, unary relations. We try to explain this viewpoint below giving first an informal motivational account, and then an account of the machinery which can be used to make the presentation rigorous. Our informal account will confuse terms and their semantics.

Jung and Tiuryn's work can be reconstructed as follows. We want to provide an invariance property of the lambda definable functions. We hope this can be expressed by a logical relation. In other words, we hope that the set of λ-definable functions is a unary logical relation. Unfortunately in the standard picture, it is not. This is because in the standard theory we can only talk about the elements of the sets which are carriers (closed terms), and there are plenty of functions which send definable elements to definable elements, but are not themselves definable. One way of looking at Jung and Tiuryn's achievement is that they found a clever way of talking about open terms. This allows us to take our operation and apply it to the definable open term "x". If the result is definable, by e say, then the operation must have been definable by $\lambda x.e$.

In Jung and Tiuryn's account the open terms appear semantically, and stratified by context. That is to say we are given a signature of (typed) variables which may appear in the operation, and the term (say of type D) appears as a function from the set of environments for that signature to D. Instead of viewing this as a function, Jung and Tiuryn view it as a tuple whose components are indexed by environments, and hence as an element of a relation on D whose arity is the set of environments. In order that a property of an open term remains true when we enlarge the environment, Jung and Tiuryn use a Kripke structure in which the worlds are the possible signatures. This means that from their perspective, what they have is a relation which is always on the set D, but whose arity varies as we move between worlds. However, another way of looking at the same structure is that the relation is always unary, but that the set on which is a relation is the set of terms definable at that world, and hence varies from world to world.

The standard way to obtain logical relations is to take a semantics in Set, and to use subsets. A theme of this paper is that this can equally well be regarded as using the standard logic of Set. This can be varied by using other categories and other logics. The technical tool we use for doing this is to interpret a logic as a fibration with structure. If the logic admits $\top$, $\wedge$, $\rightarrow$ and $\forall$, as well as substitution, then it yields a category of logical relations. The observation that these operations are all that is required to define product and exponential of logical relations should make that immediately credible, even it does not furnish a proof. These two approaches meet in the standard logic of predicates interpreted as subobjects.

In standard Kripke logical relations for the simple type hierarchy, we are given an indexing category W. The types of the hierarchy are interpreted as constant functors in the presheaf topos $[W^{op}, Set]$, so $(\Delta d)w = d$. Kripke logical relations are derived from the natural logic of the topos, that is predicates correspond to subfunctors (a subfunctor of F is a functor G such that at each world w, $Gw \subseteq Fw$). For connected W, the functor $\Delta : Set \longrightarrow [W^{op}, Set]$ is a full and faithful cartesian closed functor. This means that the simple type hierarchy on d in Set is sent by Δ to the simple type hierarchy on Δd in $[W^{op}, Set]$. So we can either regard this construction as the form of logical relations obtained from a standard logic for a non-standard set in non-standard set theory, or we can regard it as a non-standard logic for a standard set in standard set theory. The Jung-Tiuryn construction resolves this in favour of the second.

In the Jung-Tiuryn construction, W is assumed to be a concrete category, i.e., a category of sets and functions. So they have a functor $J : W \longrightarrow Set$, which they assume to be faithful (although never using that), and essentially what they do is replace the functor $\Delta : Set \longrightarrow [W^{op}, Set]$ above by the functor $\tilde{J} : \mathsf{Set} \longrightarrow [W^{op}, Set]$, sending d to the functor $\mathsf{Set}[J-, d]$.

We make two minor modifications of this. First we start with a functor, $F : D \longrightarrow C$. We turn it around to obtain the functor $\tilde{F} : C \longrightarrow [D^{op}, Set]$ sending X to the functor $C(F-, X) : D^{op} \longrightarrow Set$. We get unary logical relations by considering the subobject fibration over $[D^{op}, Set]$ (the standard logic there), and we can take the pullback of that cartesian closed fibration along $\tilde{F}$ to obtain a cartesian closed fibration over C (a non-standard logic). This involves some delicacy. Although the functor F is cartesian closed, it does not follow that $\tilde{F}$ is cartesian closed. But $\tilde{F}$ does preserve finite products, which is sufficient for the results we seek.

Next we take a slightly unexpected binary version of this. It would be normal, given a functor $K : C \longrightarrow [D^{op}, Set]$ to consider the pullback of a fibration over $[D^{op}, Set] \times [D^{op}, Set]$ along $K \times K$. But that is not what we propose. We have functors $\tilde{F}, \tilde{G} : C \longrightarrow [D^{op}, Set]$ and we shall take the pullback along $\tilde{F} \times \tilde{G}$. We do obtain the result we seek, but this is a most unusual construction. For instance, there seems no way to lift $\tilde{F} \times \tilde{G}$ to a functor from the binary subobject fibration $Sub_2(C)$ over $C \times C$ to the binary subobject fibration $Sub_2([D^{op}, Set])$

over $[D^{op}, Set] \times [D^{op}, Set]$, but that is a fundamental construction of fibrations, simply not one we need here.

Finally, one needs a little caution. Ideally, we would like the construction of our cartesian closed fibration $p : B \longrightarrow C \times C$ to be independent of the choice of F and G, with F and G only being required in order to construct $L : D \longrightarrow B$. But we cannot see how to do that: that is a particular sense in which we see the relationship with Jung and Tiuryn's work, as they have an extra parameter $J : X \longrightarrow Set$ in their work too, and it was that extra parameter that inspired us.

6 Completeness of logical relations

In this section, we consider the completeness half of the result we seek. So, given a pair $F, G : D \longrightarrow C$ of equivalent interpretations, we seek a cartesian closed fibration $p : B \longrightarrow C \times C$ and a cartesian closed functor $L : D \longrightarrow B$ lying over (F, G) such that the composite $L \cdot i$ is diagonal.

$$\begin{tikzcd}
 & & B \arrow[d, "p = (p_0, p_1)"] \\
C \arrow[r, "i"'] & D \arrow[ur, "L"] \arrow[r, "(F,G)"'] & C \times C
\end{tikzcd}$$

As mentioned in the previous section, given an arbitrary functor $F : D \longrightarrow C$, we consider the functor $\tilde{H} : C \longrightarrow [D^{op}, Set]$ that sends an object X of C to the functor $C(F-, X) : D^{op} \longrightarrow Set$. It follows from the Yoneda lemma that we have

Proposition 6. *For any $F : D \longrightarrow C$, if C has finite products, then the functor $\tilde{F} : C \longrightarrow [D^{op}, Set]$ preserves finite products.*

Also, as a standard example of a fibration [3], we have

Example 5. Let $\mathcal{C}$ be any category with finite limits. Let $Sub_2(\mathcal{C})$ have objects $(X, Y, R \to X \times Y)$ consisting of a pair (X, Y) of objects of $\mathcal{C}$ together with a subobject R of $X \times Y$, and with the evident arrows. Then the forgetful functor $Sub_2(\mathcal{C}) \longrightarrow \mathcal{C} \times \mathcal{C}$ is a fibration. If $\mathcal{C}$ is cartesian closed, the fibration is a cartesian closed fibration. It is the standard fibration of binary predicates for $\mathcal{C}$.

Proposition 7. *Given categories $\mathcal{C}$ and $\mathcal{C}'$ with finite products, a finite product preserving functor $K : \mathcal{C}' \longrightarrow \mathcal{C}$, and a finite product preserving fibration $p : \mathcal{P} \longrightarrow \mathcal{C}$, then the pullback $K^*(\mathcal{P})$ of p along K gives a finite product preserving fibration $K^*(\mathcal{P})$ over $\mathcal{C}'$.*

A proof of this result appears in Hermida's thesis [3], but it can also be verified by direct calculation.

Corollary 1. *Given $F, G : D \longrightarrow C$ where C has finite products, the pullback of $Sub_2[D^{op}, Set]$ along $\tilde{F} \times \tilde{G}$ gives a finite product preserving fibration $(\tilde{F} \times \tilde{G})^*(Sub_2[D^{op}, Set])$ over $C \times C$.*

In fact, the fibration $(\tilde{F} \times \tilde{G})^*(Sub_2[D^{op}, Set])$ is the one we want, so we shall duly denote it by B. This is essentially a *sconing* of the ordinary binary subobject fibration on the presheaf category $[D^{op}, Set]$. It is easy to prove by direct calculation, using some pullbacks in the presheaf category, that we have

Theorem 1. *Given $F, G : D \longrightarrow C$, supposing C is cartesian closed, then $B = (\tilde{F} \times \tilde{G})^*(Sub_2[D^{op}, Set])$ is a cartesian closed fibration over $C \times C$.*

Proof. The only point here that requires checking is that B is cartesian closed and that the functor to $C \times C$ preserves the cartesian closed structure. This follows from a careful analysis of what the objects of B are. Given an object (X, X') of $C \times C$, an object of B over (X, X') is a subfunctor of $C(F-, X) \times C(G-, X')$. Using cartesian closedness of $Sub_2([D^{op}, Set]$ over $[D^{op}, Set] \times [D^{op}, Set]$, given subfunctors of $C(F-, X) \times C(G-, X')$ and $C(F-, Y) \times C(G-, Y')$, one can construct a subfunctor of the exponential $[C(F-, X) \times C(G-, X'), C(F-, Y) \times C(G-, Y')]$. Finally, taking the pullback of that subfunctor along the canonical comparison map from $C(F-, [X, Y]) \times C(F-, [X', Y'])$ to $[C(F-, X) \times C(G-, X'), C(F-, Y) \times C(G-, Y')]$ gives the exponential in B that we seek.

This is a specific example of a general construction [3]. Now we have the data we require in order to state and prove our main result.

Theorem 2. *If F and G are equivalent interpretations, they are linked by a logical relation.*

Proof. Define B as in Theorem 1. It remains to define $L : D \longrightarrow B$ over (F, G), prove it is cartesian closed, and prove that $L \cdot i$ is diagonal. So for an object d of D, define Ld to be the subfunctor of $C(F-, Fd) \times C(G-, Gd)$ given by

$$Ld(c) = \{(g : Fc \longrightarrow Fd, h : Gc \longrightarrow Gd) \mid \exists f : c \longrightarrow d.\ Ff \approx g \wedge Gf \approx h\}$$

It is routine to verify that L preserves finite products; it takes considerably greater but still routine effort to verify that it preserves cartesian closed structure. It follows directly from the definition and the fact that F and G are equivalent that $L \cdot i$ is diagonal.

Combining this completeness result with the soundness result of the previous section, we have a combined soundness and completeness result as follows.

Theorem 3. *Two interpretations F and G are equivalent if and only if they are linked by a logical relation.*

References

1. Alimohamed, A.: A Characterisation of Lambda Definability in Categorical Models of Implicit Polymorphism. Theoretical Computer Science **146** (1995) 5–23.
2. Fiore, M., Plotkin, G.D.: An axiomatisation of computationally adequate domain-theoretic models of FPC. Proc LICS **94**. IEEE Press (1994) 92–102.
3. Hermida, C.A.: Fibrations, Logical Predicates and Indeterminates. Ph.D. thesis. Edinburgh (1993) available as ECS-LFCS-**93-277**.
4. Honsell, F., Sannella, D.T.: Pre-logical relations. Proc. CSL **99**. LNCS **1683**. Flum and Rodriguez-Artalejo (eds.) Springer (1999) 546–562.
5. Jung, A., Tiuryn, J.: A new characterisation of lambda definability. Proc. TLCA. LNCS **664**. Bezem and Groote (eds.) Springer (1993) 245–257.
6. Kinoshita, Y., O'Hearn, P., Power, A.J., Takeyama, M., Tennent, R.D.: An Axiomatic Approach to Binary Logical Relations with Applications to Data Refinement. Proc TACS **97**. LNCS **1281**. Abadi and Ito (eds.) Springer (1997) 191–212.
7. Lambek, J., Scott, P.J.: Introduction to higher order categorical logic. Cambridge studies in advanced mathematics. Cambridge University Press (1986)
8. Ma, Q., Reynolds, J.C.: Types, abstraction and parametric polymorphism **2**. Math. Found. of Prog. Lang. Sem. Lecture Notes in Computer Science. Springer (1991).
9. Mitchell, J.:Type systems for programming languages. Handbook of Theoretical Computer Science em B. van Leeuwen (ed). North-Holland (1990) 365–458.
10. Mitchell, J.: Foundations for programming languages. Foundations of Computing Series. MIT Press (1996).
11. Moggi, E.: Computational Lambda-calculus and Monads. Proc LICS **89**. IEEE Press (1989) 14–23.
12. Plotkin, G.D., Power, A.J., Sannella, D.T., Tennent, R.D.: Lax logical relations. (to appear in ICALP 2000).
13. Reynolds, J.C.: Types, abstraction, and parametric polymorphism. Information Processing **83** Mason (ed) North Holland (1983) 513–523.
14. Tennent, R.D.: Correctness of data representations in ALGOL-like languages. In: A Classical Mind, Essays in Honour of C.A.R. Hoare, A.W. Roscoe (ed.) Prentice-Hall (1994) 405–417.