Constructively Characterizing Fold and Unfold*

Tjark Weber[1] and James Caldwell[2]

[1] Institut für Informatik, Technische Universität München
Boltzmannstr. 3, D-85748 Garching b. München, Germany
webertj@in.tum.de

[2] Department of Computer Science, University of Wyoming
Laramie, Wyoming 82071-3315
jlc@cs.uwyo.edu

Abstract. In this paper we formally state and prove theorems characterizing when a function can be constructively reformulated using the recursion operators fold and unfold, *i.e.* given a function h, when can a function g be constructed such that $h = \mathsf{fold}\ g$ or $h = \mathsf{unfold}\ g$? These results are refinements of the classical characterization of fold and unfold given by Gibbons, Hutton and Altenkirch in [6]. The proofs presented here have been formalized in Nuprl's constructive type theory [5] and thereby yield program transformations which map a function h (accompanied by the evidence that h satisfies the required conditions), to a function g such that $h = \mathsf{fold}\ g$ or, as the case may be, $h = \mathsf{unfold}\ g$.

1 Introduction

Under the proofs-as-programs interpretation, constructive proofs of theorems relating programs yield "correct-by-construction" program transformations. In this paper we formally prove constructive theorems characterizing when a function can be formulated using the recursion operators fold and unfold, *i.e.* given a function h, when does there exist (constructively) a function g such that $h = \mathsf{fold}\ g$ or $h = \mathsf{unfold}\ g$? The proofs have been formalized in Nuprl's constructive type theory [5] and thereby yield program transformations which map a function h – accompanied by the evidence that h satisfies the required conditions – to a function g such that $h = \mathsf{fold}\ g$ or, as the case may be, $h = \mathsf{unfold}\ g$.

The results presented here are refinements of the classical characterization of fold / unfold given by Gibbons, Hutton and Altenkirch in [6]. As they remark, their characterization is set theoretic and makes essential use of classical logic and the Axiom of Choice. A constructive characterization of fold was given by Weber in [15] and a counter-example showing that indeed, under the characterization given in [6], there are constructive functions h that can be written in the form $\mathsf{fold}\ g$ where g is necessarily incomputable. We extend those results here and

* This work was supported by NSF grant CCR-9985239, a DoD Multidisciplinary University Research Initiative (MURI) program administered by the Office of Naval Research under grant N00014-01-1-0765, and by the PhD program Logic in Computer Science of the German Research Foundation.

present a constructive characterization for both fold and unfold. Following [6] our results are presented in the context of a category-theoretic framework, also formalized in Nuprl.

In the next section we describe the definitions of the Nuprl formalization of category theory required later in the paper resisting elaboration. We do not state or prove any theorems in this section. A description of the Nuprl formalization of category theory can be found in [15]. In following sections we define catamorphisms and anamorphisms and, relying on their universal property, give formal Nuprl definitions of fold and unfold. We present the statements of the theorems which classically characterize fold and unfold from [6]. It turns out that one direction of the classical theorems is constructively provable. For the other direction, we refine the conditions on the antecedents to obtain characterizations of fold and unfold which hold constructively. The constructive content of the proof of these theorems are the desired program transformations.

2 Category Theory in Nuprl

The Nuprl type theory and proof system have previously been described in this conference [4], a recent and comprehensive reference for Nuprl's constructive type theory is available on-line [1]. In short, Nuprl draws heavily on Martin-Löf type theory [12], which uses an open-ended sequence of universes $\mathbb{U}_1$, $\mathbb{U}_2$, $\mathbb{U}_3$, ... to stratify the concept of type.

The formal Nuprl definition of the type *category* (up through universe level i) is shown in Fig. 1. This definition follows the standard definition, as found in say [10]. In the definition: `Obj` is the type of objects in the category; `A` is the type of arrows; `dom` and `cod` are the functions mapping arrows to their domains and codomains; `o` is the type of the composition operator (which is constrained to be defined only on arrows whose domains and codomains align properly and is associative); and the final component of the product, `id`, specifies the function which maps objects to arrows preserving the unit law.

```
Cat{i} ≝
  Obj:𝕌ᵢ
  × A:𝕌ᵢ
  × dom:(A → Obj)
  × cod:(A → Obj)
  × o:{o:(g:A → f:{f:A| cod f = dom g} →
                     {h:A| dom h = dom f ∧ cod h = cod g}) |
       ∀f,g,h:A. cod f = dom g ∧ cod g = dom h ⇒
                       (h o g) o f = h o (g o f)
      }
  × {id:(p:Obj → {f:A| dom f = p ∧ cod f = p}) |
      ∀f:A. (id(cod f)) o f = f ∧ f o (id(dom f)) = f}
```

Fig. 1. Abstraction: `category`

For a category `C` we use selectors `C_Obj, C_Arr, C_dom, C_cod, C_op` and `C_id` to refer to the components of `C`.

The analog of the large category of sets in our type theoretic formulation is the category of types whose universe level is bounded by some $i \in \mathbb{N}$. The arrows in this category are triples of the form $\langle A, B, f\rangle$ where $A, B \in \mathbb{U}_i$ and $f \in A \to B$. The category of types is defined in Fig. 2.

```
large_category{i} ≝
  <𝕌ᵢ,
   (A:𝕌ᵢ × B:𝕌ᵢ × (A → B)),
   λf. f.1,
   λf. f.2.1,
   λg,f. <f.1, g.2.1, g.2.2 o f.2.2>,
   λp. <p, p, λx.x>>
```

Fig. 2. Abstraction: `large_category`

The well-formedness goal is stated as: `large_category{i} ∈ Cat{i'}`, *i.e.* it says that the category of types below universe level i inhabits the type category at level $i + 1$.

A miscellany of defined notions used later in the paper are displayed in Fig. 3.

```
C-composable(f,g)  ≝  C_cod f = C_dom g
Mor[C](p,q)        ≝  {f:C_Arr| C_dom f = p ∧ C_cod f = q}
C-initial(p)       ≝  ∀q:C_Obj. ∃!f:C_Arr. C_dom f = p ∧ C_cod f = q
C-terminal(p)      ≝  ∀q:C_Obj. ∃!f:C_Arr. C_dom f = q ∧ C_cod f = p
```

Fig. 3. Abstractions: `composable`, `morphism`, `initial` and `terminal`

Functors are arrows between categories. A functor from $\mathcal{C}$ to $\mathcal{D}$, where $\mathcal{C}$ and $\mathcal{D}$ are categories, maps objects in $\mathcal{C}$ to objects in $\mathcal{D}$ and arrows in $\mathcal{C}$ to arrows in $\mathcal{D}$ such that these maps preserve structure, *i.e.* they are compatible with the categories' domain and codomain operators, preserve identity elements and respect composition of arrows. The formal definition is given in Fig. 4.

```
Functor{i}(C,D) ≝
  {C:Cat{i}}
  × {D:Cat{i}}
  × O:(C_Obj → D_Obj)
  × {M:C_Arr → D_Arr|
       (∀f:C_Arr. D_dom (M f) = O (C_dom f)
                ∧ D_cod (M f) = O (C_cod f))
       c∧ ((∀f:C_Arr. ∀g:{g:C_Arr| C_dom g = C_cod f}
             M (g C_op f) = (M g) D_op (M f))
       ∧ (∀p:C_Obj. M (C_id p) = D_id (O p)))}
F_dom ≝F.1
F_cod ≝F.2.1
```

Fig. 4. Abstractions: `functor`, `functor_dom` and `functor_cod`

Given a category $\mathcal{C}$ and a functor $F : \mathcal{C} \to \mathcal{C}$, an *algebra* over F is a pair $\langle A, f\rangle$, where A is an object and $f : FA \to A$ is an arrow in $\mathcal{C}$. The formal definitions are given in Fig. 5.

```
Algebra(F) ≝
  p:F_dom_Obj × {f:F_dom_Arr| F_dom_dom f = F_O p ∧ F_dom_cod f = p}
Hom(F) ≝
  A:Algebra(F)
   × B:Algebra(F)
   × {f:F_dom_Arr| (F_dom_dom f = A_obj) c∧ (F_dom_cod f = B_obj)
                    c∧ (f F_dom_op A_arr = B_arr F_dom_op (F_M f))}
algebra_category(F) ≝<Algebra(F), Hom(F), λh.h_dom, λh.h_cod,
                       λh,g.h o_hom[F] g, λA.id_hom[F](A)>
```

Fig. 5. Abstractions: `algebra`, `homomorphism` and `algebra_category`

A *coalgebra* is a pair $\langle A', f'\rangle$, where A' is an object and $f' : A' \to FA'$ is an arrow in $\mathcal{C}$. Thus, coalgebras are algebras over the dual category. The formal definitions are given in Fig. 6.

```
Coalgebra(F) ≝
  p:F_dom_Obj × {f:F_dom_Arr| F_dom_dom f = p ∧ F_dom_cod f = F_O p}
Cohom(F) ≝
  A:Coalgebra(F)
    × B:Coalgebra(F)
    × {f:F_dom_Arr| (F_dom_dom f = A_obj) c∧ (F_dom_cod f = B_obj)
                     c∧ ((F_M f) F_dom_op A_arr = B_arr F_dom_op f)}
coalgebra_category(F) ≝<Coalgebra(F), Cohom(F), λh.h_dom, λh.h_cod,
                         λh,g.h o_cohom[F] g, λA.id_cohom[F](A)>
```

Fig. 6. Abstractions: `coalgebra`, `cohomomorphism` and `coalgebra_category`

3 Catamorphisms and Anamorphisms

Catamorphisms ('folds') and anamorphisms ('unfolds') can be formalized as certain arrows in the category of algebras and in the category of coalgebras, respectively. Significantly, they serve as a basis for a transformational approach to functional programming [3] and a wide variety of transformations, optimizations and proof techniques are known for algorithms that are expressed as combinations of folds and unfolds [14, 2, 8, 7, 9, 15].

Catamorphisms are homomorphisms from an initial algebra in the category of algebras, *anamorphisms* are defined as cohomomorphisms to a terminal coalgebra in the category of coalgebras.

An algebra $\langle \mu F, in\rangle$ is *initial* if and only if it is an initial object (see Fig. 3) in the category of algebras; that is, for every algebra $\langle A, f\rangle$, there exists a unique homomorphism $h : \langle \mu F, in\rangle \to \langle A, f\rangle$. A coalgebra $\langle \nu F, out\rangle$ is *terminal* if and only if it is a terminal object in the category of coalgebras; *i.e.*, for every coalgebra $\langle A, f\rangle$, there exists a unique cohomomorphism $h : \langle A, f\rangle \to \langle \nu F, out\rangle$.

Definition 1 (fold) *Suppose $\mathcal{C}$ is a category, $F : \mathcal{C} \to \mathcal{C}$ is a functor and $\langle \mu F, in\rangle$ is an initial algebra. Then for every algebra $\langle A, f\rangle$, $\mathsf{fold}\, f$ is defined as the unique homomorphism from $\langle \mu F, in\rangle$ to $\langle A, f\rangle$.*

$$\begin{tikzcd}
F(\mu F) \arrow[r, "F(\mathsf{fold}\, f)"] \arrow[d, "in"'] & FA \arrow[d, "f"] \\
\mu F \arrow[r, dotted, "\mathsf{fold}\, f"'] & A
\end{tikzcd}$$

Fig. 7. $(\mathsf{fold}\, f) \cdot in = f \cdot F(\mathsf{fold}\, f)$

We say an arrow h is a *catamorphism* if and only if it can be written as $\mathsf{fold}\, f$ for some arrow f.

Definition 2 (unfold) *Suppose $\mathcal{C}$ is a category, $F : \mathcal{C} \to \mathcal{C}$ is a functor and $\langle \nu F, out\rangle$ is a terminal coalgebra. Then for every coalgebra $\langle A, f\rangle$, $\mathsf{unfold}\, f$ is defined as the unique cohomomorphism from $\langle A, f\rangle$ to $\langle \nu F, out\rangle$.*

$$\begin{tikzcd}
A \arrow[r, dotted, "\mathsf{unfold}\, f"] \arrow[d, "f"'] & \nu F \arrow[d, "out"] \\
FA \arrow[r, "F(\mathsf{unfold}\, f)"'] & F(\nu F)
\end{tikzcd}$$

Fig. 8. $F(\mathsf{unfold}\, f) \cdot f = out \cdot (\mathsf{unfold}\, f)$

Figure 8 illustrates this situation. We say an arrow h is an *anamorphism* if and only if it can be written as $\mathsf{unfold}\, f$ for some arrow f. These definitions imply the following *universal properties* for fold and unfold [11].

Theorem 1 (Universal Property: fold). *Let $\mathcal{C}$ be a category, $F : \mathcal{C} \to \mathcal{C}$ a functor and $\langle \mu F, in\rangle$ an initial algebra. Furthermore, suppose that $\langle A, f\rangle$ is an algebra and that $h : \mu F \to A$. Then*

$$h = \mathsf{fold}\, f \iff h \cdot in = f \cdot Fh.$$

Theorem 2 (Universal Property: unfold). *Let $\mathcal{C}$ be a category, $F : \mathcal{C} \to \mathcal{C}$ a functor and $\langle \nu F, out\rangle$ a terminal coalgebra. Furthermore, suppose that $\langle A, f\rangle$ is a coalgebra and that $h : A \to \nu F$. Then*

$$h = \mathsf{unfold}\, f \iff Fh \cdot f = out \cdot h.$$

Based on their universal properties, formalize fold and unfold as relations (shown in Fig. 9). The well-formedness theorems state that they inhabit $\mathbb{P}$, Nuprl's type of propositions. We remark that h is unique when $h = \mathsf{unfold}\, f$ or $h = \mathsf{fold}\, f$.

```
h=fold[C,F,I](f) ≝(F_dom_dom h = I_obj) c∧ (F_dom_cod h = f_obj)
                   c∧ (h F_dom_op I_arr = f_arr F_dom_op (F_M h))
h=unfold[C,F,T](f) ≝(F_dom_dom h = f_obj) c∧ (F_dom_cod h = T_obj)
                   c∧ ((F_M h) F_dom_op f_arr = T_arr F_dom_op h)
```

Fig. 9. Abstractions: `fold` and `unfold`

4 When is an Arrow a Catamorphism or an Anamorphism?

The universal properties for fold and unfold provide technically complete answers to this question. An arrow $h : \mu F \to A$ is a catamorphism if and only if $h \cdot in = f \cdot Fh$ for some arrow $f : FA \to A$. However, usually only the arrow h is given—how would we know if an arrow f exists such that the above equation holds? And more importantly, how would we construct f from h? Dually, an arrow $h : A \to \nu F$ is an anamorphism if and only if $Fh \cdot f = out \cdot h$ for some arrow $f : A \to FA$. Again, to show that f exists or methods to construct it are not given.

E. Meijer, M. Fokkinga, and R. Paterson [13] give the following results regarding left and right invertible arrows.

Definition 3 (Left and Right Invertible) *Let $\mathcal{C}$ be a category and f an arrow in $\mathcal{C}$.*

1.) *We say f is* left-invertible *(in $\mathcal{C}$) if and only if there exists an arrow g in $\mathcal{C}$ such that $g \cdot f = id(dom(f))$.*
2.) *We say f is* right-invertible *(in $\mathcal{C}$) if and only if there exists an arrow g in $\mathcal{C}$ such that $f \cdot g = id(cod(f))$.*

The corresponding Nuprl abstractions are shown in Fig. 10. Their well-formedness theorems simply state that these abstractions are propositions.

```
left-invertible[C](f) ≝
  ∃g:{g:C_Arr| C-composable(f,g)} . g C_op f = C_id (C_dom f)
right-invertible[C](f) ≝
  ∃g:{g:C_Arr| C-composable(g,f)} . f C_op g = C_id (C_cod f)
```

Fig. 10. Abstractions: `left_invertible` and `right_invertible`

The following theorems provide tools to show when f exists.

Theorem 3. *If $\mathcal{C}$ is a category, $F : \mathcal{C} \to \mathcal{C}$ is a functor with an initial algebra $\langle \mu F, in \rangle$, and $h : \mu F \to A$ is a left-invertible arrow in C, then, for some arrow $f : FA \to A$, $h = \mathsf{fold}\, f$.*

Theorem 4. *If $\mathcal{C}$ is a category, $F : \mathcal{C} \to \mathcal{C}$ is a functor with a terminal coalgebra $\langle \nu F, out \rangle$, and $h : A \to \nu F$ is a right-invertible arrow in C, then, for some arrow $f : A \to FA$, $h = \mathsf{unfold}\, f$.*

The Nuprl theorems formalizing these results are shown in Fig. 11. Proofs in Nuprl are created in an interactive fashion. In each proof step, instances of (one or more) proof rules are chosen by the user and applied to the current sequent. Both theorems above are proved in about 70 steps each.

```
∀C:Cat{i}. ∀F:Functor{i}(C,C).
 ∀I:{I:Algebra(F)| algebra_category(F)-initial(I)} .
  ∀h:{h:F_dom_Arr| F_dom_dom h = I_obj} .
   left-invertible[F_dom](h) ⇒ (∃f:Algebra(F). h=fold[C,F,I](f) )
∀C:Cat{i}. ∀F:Functor{i}(C,C)
 ∀T:{T:Coalgebra(F)| coalgebra_category(F)-terminal(T)}
  ∀h:{h:F_dom_Arr| F_dom_cod h = T_obj}
   right-invertible[F_dom](h) ⇒ ∃f:Coalgebra(F). h=unfold[C,F,T](f)
```

Fig. 11. Thms: `left_invertible_implies_fold` `right_invertible_implies_unfold`

Figure 12 shows the extract[3] of the proof of `right_invertible_implies_unfold`. We can clearly see the witness term in Nuprl notation: The witness term is given by the coalgebra `<F_dom_dom h, (F_M g) F_dom_op (T_arr F_dom_op h)>`. A similar extract results from the proof of the theorem `left_invertible_implies_fold`.

```
λC,F,T,h,p.
 let <g,_> = p in
   <<F_dom_dom h, (F_M g) F_dom_op (T_arr F_dom_op h)>, Ax, Ax, Ax>
```

Fig. 12. Simplified Extract of `right_invertible_implies_unfold`

5 Classically Characterizing fold and unfold

For the special case of the category $\mathcal{SET}$, with sets as objects and functions as arrows, J. Gibbons, G. Hutton, and T. Altenkirch [6] proved the following theorems characterizing when an arrow is a catamorphism or an anamorphism.

Theorem 5 (Gibbons, Hutton, Altenkirch: fold). *Let $F : \mathcal{SET} \to \mathcal{SET}$ be a functor with an initial algebra $\langle \mu F, in \rangle$, A be a set, and $h : \mu F \to A$. Then $(\exists g : FA \to A.\quad h = \mathsf{fold}\, g) \iff \mathsf{ker}(Fh) \subseteq \mathsf{ker}(h \cdot in)$.*

[3] The extract has been simplified by unfolding definitions, performing β-reductions and α-renaming selected variables to make the code more readable.

Here, $\mathsf{ker}\ f$, the *kernel* of a function $f : A \to B$, is defined as a binary relation on A containing all pairs of elements in A that are mapped to the same element in B. It is formalized in Fig. 13.

We remark here that this theorem of classical set theory is too strong in the following sense: there exists a function h such that h is computable and such that $\mathsf{ker}\,(Fh) \subseteq \mathsf{ker}\,(h \cdot in)$ but where g, which exists by Thm. 5, is necessarily incomputable [15].

The following theorem characterizes the dual unfold.

Theorem 6 (Gibbons, Hutton, Altenkirch: unfold). *Let $F : \mathcal{SET} \to \mathcal{SET}$ be a functor with a terminal coalgebra $\langle \nu F, out\rangle$, A be a set, and $h : A \to \nu F$. Then $(\exists g : A \to FA.\quad h = \mathsf{unfold}\, g) \iff \mathsf{img}(out \cdot h) \subseteq \mathsf{img}(Fh)$.*

The *image* of a function $f : A \to B$, $\mathsf{img}\, f$, is the dual notion to the kernel of f (see Fig. 13).

```
ker[A,B] f ≝{aa:A × A| f aa.1 = f aa.2}
img[A,B] f ≝{b:B| ∃a:A. b = f a}
```

Fig. 13. Abstractions: `kernel` and `image`

6 A Constructive Characterization of fold and unfold

Translating the statements of Thms. 5 and 6 so that the category $\mathcal{SET}$ is replaced by the large category of types result in theorems that are constructively provable in the ($\Rightarrow$) direction [15]. However, the ($\Leftarrow$) direction contains the computationally interesting parts of these theorems; it claims existence for the function g we are interested in.

6.1 Characterizing fold

Analyzing the proof of Thm. 5 led us to identify additional constraints that in fact allow a constructive proof of a modified version of the ($\Leftarrow$) direction. Before we state these conditions, we must address an issue that is not raised by differences between classical and constructive mathematics, but by the inherent differences between set theory and type theory.

Thus far, while considering the constructive interpretation of the classical results we have interpreted types *mutatis mutandis* as sets. Up to this point this informal practice has proved harmless, but at this point our naïve identification of sets and types fails. Consider the analogue of the empty set, *i.e.* types having no inhabitants. Equality on types in Nuprl is not extensional as it is for sets. Hence, unlike set theory, where every set containing no elements is identified with $\emptyset$, there is no canonical representative for the empty type; *e.g.* neither *Void* nor $\{x : \mathbb{Z}|\ x < x\}$ are inhabited and yet they are distinguished as types. The identification of empty sets with *the* empty set is a crucial step in the ($\Leftarrow$) direction of the classical proof.

Here is the statement of the refined theorem (still in terms of the category $\mathcal{SET}$) corresponding to the ($\Leftarrow$) part of Thm. 5.

Theorem 7. *Let $F : \mathcal{SET} \to \mathcal{SET}$ be a functor with an initial algebra $\langle \mu F, in \rangle$, and let A be a set such that we can decide whether A is empty, and $h : \mu F \to A$. Furthermore, suppose that for every $b \in FA$ we can decide whether $b = (Fh)(a)$ for some $a \in F(\mu F)$. Then $(\exists g \colon FA \to A. h = \mathsf{fold}\, g) \Longleftarrow \ker(Fh) \subseteq \ker(h \cdot in)$.*

Figure 14 shows a type-theoretic formalization of this theorem in Nuprl. `Dec(P)` is used to abbreviate `P ∨ ¬P`. The proof depends on two lemmata, namely that the inclusion of kernels implies the existence of postfactors, and that the existence of a function $h : \mu F \to A$ implies the existence of a function $g : FA \to A$. We say $g : B \to C$ is a *postfactor* of $f : A \to B$ for $h : A \to C$ if and only if $h = g \cdot f$. We will use the notation $A \to B \neq \emptyset$ to mean that there is a function inhabiting $A \to B$.

We state and prove the former lemma first.

```
∀F:Functor{i'}(large_category{i},large_category{i}).
 ∀I:{I:Algebra(F)| algebra_category(F)-initial(I)} .
  ∀A:large_category{i}_Obj. ∀h:Mor[large_category{i}](I_obj,A).
   Dec(A) ⇒ (∀b:F_O A. Dec(∃a:F_O I_obj. b = (F_M h).2.2 a)) ⇒
     ((∃g:Algebra(F). h=fold[large_category{i},F,I](g) )
        ⇐ ker[F_O I_obj,large_category{i}_cod (F_M h)] (F_M h).2.2
            ⊆ ker[F_O I_obj,large_category{i}_cod (h F_dom_op I_arr)]
                 (h F_dom_op I_arr).2.2)
```

Fig. 14. Theorem: `kernel_inclusion_implies_fold`

Lemma 1. *Let $f : A \to B$ and $h : A \to C$. Furthermore, suppose we can decide whether C is empty, and for every $b \in B$ we can decide whether $b = f(a)$ for some $a \in A$. Then $(\exists g : B \to C. \quad h = g \cdot f) \Longleftarrow (\ker f \subseteq \ker h \wedge B \to C \neq \emptyset)$.*

Proof. Assume $\ker f \subseteq \ker h$ and $B \to C \neq \emptyset$.

If $C = \emptyset$, then $B = \emptyset$ since $B \to C \neq \emptyset$, and $A = \emptyset$ since $f : A \to B$. Therefore $f = h = id(\emptyset)$, and if we choose $g = id(\emptyset)$, clearly $g : B \to C$ and $h = g \cdot f$.

If $C \neq \emptyset$, let c be an arbitrary element in C. Let $choice : \{b \in B \mid \exists a \in A.\ b = f(a)\} \to A$ be a function with $f(choice(b)) = b$ for all $b \in \{b \in B \mid \exists a \in A.\ b = f(a)\}$. [4] For $b \in B$ define $g(b) \in C$ as follows: If $b = f(a)$ for some $a \in A$, then $g(b) = h(choice(b))$. Otherwise, $g(b) = c$.

Now let $a \in A$. Since $f(choice(f(a))) = f(a)$ by definition of *choice*, we have $(choice(f(a)), a) \in \ker f \subseteq \ker h$. Hence $g(f(a)) = h(choice(f(a))) = h(a)$, and therefore $h = g \cdot f$. □

[4] To prove that such a function *choice* exists, we use the Axiom of Choice which is provable in constructive type theory [12] and is a theorem in the Nuprl standard library.

```
∀A,B,C:𝕌ᵢ. ∀f:A → B. ∀h:A → C.
 Dec(C) ⇒ (∀b:B. Dec(∃a:A. b = f a)) ⇒
 ((∃g:B → C. h = g o f) ⇐ ker[A,B]f ⊆ ker[A,C]h ∧ B → C)
```

Fig. 15. Theorem: `kernel_inclusion_implies_postfactor`

To give a constructive proof that the inclusion of kernels implies the existence of postfactors, we made two additional assumptions compared to the statement of this lemma in [6]: *i.)* that we can decide whether the codomain of h is empty, and *ii.)* that we can decide whether an element in the codomain of f is in the image of f. The Nuprl theorem `kernel_inclusion_implies_postfactor` is shown in Figure 15. The formal proof is about 43 steps long.

Figure 16 shows a “lifted” version of the lemma for arrows in the category of types. Despite the use of the original lemma in the proof of the lifted version, the proof is about 71 steps long.

```
∀A,B,C:large_category{i}_Obj. ∀f:Mor[large_category{i}](A,B).
 ∀h:Mor[large_category{i}](A,C).
  Dec(C) ⇒ (∀b:B. Dec(∃a:A. b = f.2.2 a)) ⇒
    ((∃g:Mor[large_category{i}](B,C). h = g large_category{i}_op f)
       ⇐ ker[A,B]f.2.2 ⊆ ker[A,C]h.2.2 ∧ Mor[large_category{i}](B,C))
```

Fig. 16. Theorem: `kernel_inclusion_implies_postfactor_cat`

The second lemma required for the proof of Thm. 7 is stated below.

Lemma 2. *If $F : \mathcal{SET} \to \mathcal{SET}$ is a functor with an initial algebra $\langle \mu F, in \rangle$, and A is a set such that we can decide whether A is empty, then*

$$\mu F \to A \neq \emptyset \implies FA \to A \neq \emptyset.$$

Proof. If $A \neq \emptyset$, then trivially $FA \to A \neq \emptyset$.

If $A = \emptyset$, then the embedding $g : A \hookrightarrow \mu F$ is a function from A to μF. Thus $Fg : FA \to F(\mu F)$ by the properties of functors. Hence $h \cdot in \cdot Fg : FA \to A$.

Therefore $FA \to A \neq \emptyset$ in either case. □

$$\begin{tikzcd}
F(\mu F) \arrow[d, "in"'] & FA \arrow[l, dashed, bend right, "Fg"'] \arrow[d, dotted, "f"] \\
\mu F \arrow[r, "h"'] & A \arrow[l, dashed, bend right, "g"']
\end{tikzcd}$$

Fig. 17. $\mu F \to A \neq \emptyset \implies FA \to A \neq \emptyset$.

Figure 17 illustrates the situation: Given a function $h : \mu F \to A$, we can find a function $f : FA \to A$. The functions $g : A \to \mu F$ and $Fg : FA \to F(\mu F)$ are needed only in the case $A = \emptyset$. If $A \neq \emptyset$, they may not exist—but we can construct a function $f : FA \to A$ directly then. Note that the lemma is not true for arbitrary categories. The proof of the lemma given above is different from the proof that was given in [6],[5] but the theorem `hom_fun_implies_algebra_fun` (which is shown in Figure 18) is proved along the same lines. The formal proof is about 49 steps long.

```
∀F:Functor{i'}(large_category{i},large_category{i}).
  ∀I:{I:Algebra(F)| algebra_category(F)-initial(I)} .
   ∀A:large_category{i}_Obj. Dec(A) ⇒
    Mor[large_category{i}](I_obj,A) ⇒Mor[large_category{i}](F_O A,A)
```

Fig. 18. Theorem: `hom_fun_implies_algebra_fun`

We are now ready to prove Theorem 7.

Proof.

$$
\begin{array}{ll}
& \ker(Fh) \subseteq \ker(h \cdot \mathit{in}) \\
\Longleftrightarrow & \quad \{ \text{ Lemma 2, } h : \mu F \to A \ \} \\
& \ker(Fh) \subseteq \ker(h \cdot \mathit{in}) \wedge FA \to A \neq \emptyset \\
\Longrightarrow & \quad \{ \text{ Lemma 1 } \} \\
& \exists g : FA \to A. \quad h \cdot \mathit{in} = g \cdot Fh \\
\Longleftrightarrow & \quad \{ \text{ universal property } \} \\
& \exists g : FA \to A. \quad h = \mathsf{fold}\, g.
\end{array}
$$

□

Clearly we can decide whether an element in FA is in the image of Fh when Fh is surjective (onto). We will show that Fh is surjective if h is. Therefore every surjective function that satisfies the condition of kernel inclusion is a catamorphism if we can decide whether its codomain A is empty.[6] We could relatively easily prove this as a corollary to Theorem 7. Closer inspection of the proof of Theorem 7 however shows that when h is surjective, we do not need the additional assumption that we can decide whether A is empty.

Theorem 8. *Suppose $F : \mathcal{SET} \to \mathcal{SET}$ is a functor with an initial algebra $\langle \mu F, \mathit{in} \rangle$, and $h : \mu F \to A$ is surjective. Then*

$$(\exists g : FA \to A. \quad h = \mathit{fold}\, g) \Longleftarrow \ker(Fh) \subseteq \ker(h \cdot \mathit{in}).$$

We first prove that a function is surjective if and only if it is right-invertible in $\mathcal{SET}$.

Lemma 3. *Suppose $f : A \to B$. Then*

$$f \text{ is surjective} \Longleftrightarrow f \text{ is right-invertible in } \mathcal{SET}.$$

[5] The differences between our proofs can be attributed to the empty type issue mentioned earlier, but also because we avoided the form of contrapositive used there, $(\neg p \Rightarrow \neg q) \Rightarrow (q \Rightarrow p)$, which is not constructively valid

[6] Note that every injective (one-to-one) function is a catamorphism by Theorem 3.

Proof. For the ($\Rightarrow$) direction, suppose f is surjective. Then there exists a function $g : B \to A$ such that $f(g(b)) = b$ for all $b \in B$ (by the Axiom of Choice). Hence $f \cdot g = id(B)$, so f is right-invertible.

For the ($\Leftarrow$) direction, suppose f is right-invertible in $\mathcal{SET}$. Then $f \cdot g = id(B)$ for some function $g : B \to A$. Now let $b \in B$. Then $f(g(b)) = (f \cdot g)(b) = (id(B))(b) = b$. Therefore f is surjective. □

Figure 19 shows a formalization of the lemma in Nuprl. The formal proof is about 33 steps long and makes use of the `ax_choice` lemma from the Nuprl standard library.

```
∀A,B:𝕌ᵢ. ∀f:A → B.
   Surj(A;B;f) ⟺ right-invertible[large_category{i}](<A, B, f>)
```

Fig. 19. Theorem: `surjective_iff_right_invertible`

We also state and prove a lifted version of the lemma for arrows in the category of types. This lifted version is shown in Figure 20. Lifting the lemma requires about 11 proof steps using the lemma `surjective_iff_right_invertible`.

```
∀f:large_category{i}_Arr
   Surj(large_category{i}_dom f;large_category{i}_cod f;f.2.2)
   ⟺ right-invertible[large_category{i}](f)
```

Fig. 20. Theorem `surjective_iff_right_invertible_cat`

We now prove a lemma similar to Lemma 1, but for surjective functions.

Lemma 4. *Suppose $f : A \to B$ is surjective, and suppose $h : A \to C$. Then*

$$(\exists g : B \to C. \quad h = g \cdot f) \Longleftarrow \ker f \subseteq \ker h.$$

Proof. Assume $\ker f \subseteq \ker h$.

Let $choice : B \to A$ be a function with $f(choice(b)) = b$ for all $b \in B$ (such a function $choice$ exists by the Axiom of Choice since f is surjective). Define $g : B \to C$ by $g(b) = h(choice(b))$ for every $b \in B$.

Now $h = g \cdot f$ by construction of g: Let $a \in A$. Since $f(choice(f(a))) = f(a)$ by definition of $choice$, $(choice(f(a)), a) \in \ker f \subseteq \ker h$. Therefore $g(f(a)) = h(choice(f(a))) = h(a)$. □

Figure 21 shows a formalization of this lemma in Nuprl. The formal proof requires about 14 steps. It is similar to the proof of `kernel_inclusion_implies_postfactor`, but slightly simpler—just like the informal proof.

```
∀A,B,C:𝕌ᵢ. ∀f:A → B. ∀h:A → C. Surj(A;B;f) ⇒
  ((∃g:B → C. h = g o f) ⇐ ker[A,B] f ⊆ ker[A,C] h)
```

Fig. 21. Theorem: `kernel_inclusion_implies_postfactor_surjective`

As for the `kernel_inclusion_implies_postfactor` lemma above, we prove a lifted version of this lemma for arrows in the category of types. The lifted version is shown in Figure 22. Its proof is similar to the proof of the lifted lemma for functions with a decidable image (see Figure 16) and requires about 47 steps.

```
∀A,B,C:large_category{i}_Obj. ∀f:Mor[large_category{i}](A,B).
 ∀h:Mor[large_category{i}](A,C). Surj(A;B;f.2.2) ⇒
  ((∃g:Mor[large_category{i}](B,C). h = g large_category{i}_op f)
     ⇐ ker[A,B] f.2.2 ⊆ ker[A,C] h.2.2)
```

Fig. 22. Theorem: `kernel_inclusion_implies_postfactor_surjective_cat`

Using the two Lemmata 3 and 4, we can now prove Theorem 8.

Proof. We first show that $Fh : F(\mu F) \to FA$ is surjective. Since h is surjective, h is right-invertible by Lemma 3. Let $g : A \to \mu F$ be a function with $h \cdot g = id(A)$. Then

$$\begin{array}{cl} & Fh \cdot Fg \\ = & \quad \{ \text{ functors } \} \\ & F(h \cdot g) \\ = & \quad \{ \text{ assumption } \} \\ & F(id(A)) \\ = & \quad \{ \text{ functors } \} \\ & id(FA). \end{array}$$

Hence Fh is right-invertible, and therefore surjective (again by Lemma 3). Now

$$\begin{array}{cl} & \ker(Fh) \subseteq \ker(h \cdot in) \\ \Longrightarrow & \quad \{ \text{ Lemma 4 } \} \\ & \exists g : FA \to A. \quad h \cdot in = g \cdot Fh \\ \Longleftrightarrow & \quad \{ \text{ universal property } \} \\ & \exists g : FA \to A. \quad h = \mathsf{fold}\, g \end{array}$$

completing the proof. □

See Figure 23 for a statement of this theorem in Nuprl. We use the lemma `kernel_inclusion_implies_postfactor_surjective_cat` to prove the existence of g, and `surjective_iff_right_invertible_cat` to prove that Fh is surjective. Altogether the formal proof requires about 145 steps.

We now have two simple conditions for when a constructive function h that satisfies the condition of kernel inclusion is a catamorphism: h is a catamorphism if the image of Fh is decidable and we can decide whether the codomain of h is empty, and h is a catamorphism if h is surjective.

6.2 A small example

Embedded in the constructive proof of Thm. 7 is an algorithm to compute a function g such that $h = \mathsf{fold}\, g$. As an example, we want to apply this algorithm

```
∀F:Functor{i'}(large_category{i},large_category{i}).
 ∀I:{I:Algebra(F)| algebra_category(F)-initial(I)} .
  ∀A:large_category{i}_Obj. ∀h:Mor[large_category{i}](I_obj,A).
   Surj(I_obj;A;h.2.2) ⇒
     ((∃g:Algebra(F). h=fold[large_category{i},F,I](g) )
        ⇐ ker[F_O I_obj,large_category{i}_cod (F_M h)] (F_M h).2.2
            ⊆ ker[F_O I_obj,large_category{i}_cod (h F_dom_op I_arr)]
                 (h F_dom_op I_arr).2.2)
```

Fig. 23. Theorem: `kernel_inclusion_implies_fold_surjective`

to the function `all`, defined by `all p L = and (map p L)`. Here L is a list over some type T, and $p : T \to \mathbb{B}$. This function computes whether all elements in L satisfy the predicate p. To do so however, the implementation first iterates over L to compute an intermediate list of boolean values, and then it iterates over the list of booleans to compute their conjunction. Writing `all` directly as a catamorphism would eliminate the need for an intermediate list.

Before we can prove that `all` can be written as a catamorphism, we have to show that $List(T)$ is the object of an initial algebra. Consider the functor $\mathcal{L}_T : \mathcal{SET} \to \mathcal{SET}$, defined by $\mathcal{L}_T(A) = \mathbf{1} + (T \times A)$ and $\mathcal{L}_T(f) = id(\mathbf{1}) + (id(T) \times f)$. Formally verifying that this is in fact a functor takes about 54 proof steps in Nuprl. This functor has an initial algebra $(\mu\mathcal{L}_T, in) = (List(T), nil + cons)$. To verify initiality, we have to show that for every other algebra (A, f) there exists a unique homomorphism h from $(List(T), nil + cons)$ to (A, f). Since h is a homomorphism, $h([]) = f(inl\ \cdot)$ and $h(u :: v) = f(inr\ (u, h(v)))$ for all $u \in T$, $v \in List(T)$. Both that h is a homomorphism and that h is unique can then be proved by structural induction on lists. The formal proof is quite technical, and complicated by our inevitable formalization of algebras, homomorphisms and arrows in the category of types as tuples. With approximately 211 proof steps, it is the longest proof in this paper. About 140 of those steps are required only to show uniqueness of h. However, initiality only needs to be proven once for each data-type. Having proven initiality of $List(T)$, we can treat any list-consuming function, not just `all`.

Using the `kernel_inclusion_implies_fold` theorem, we can now prove that the composition of `map` and `and` is a catamorphism. We need just one more assumption: that we can decide for all $b \in \mathbb{B}$ whether there exists a list $L \in List(T)$ with $b = and(map(p; L))$. Since $and(map(p; [])) =$ true, it is sufficient if we can decide whether $p(t) =$ false for some $t \in T$. (If there is such a t, false $= and(map(p; t :: []))$. Otherwise $and(map(p; L)) =$ true for all $L \in List(T)$. This argument is reflected in the structure of the resulting program.) Figure 24 shows the Nuprl theorem `list_and_2_map_is_fold`.

```
* THM list_and_2_map_is_fold
∀T:𝕌. ∀p:T → 𝔹.
Dec(∃t:T. p t = false)
⇒ (∃g:Algebra(ListF{i}(T))
    <T List, 𝔹, λL.∧b(map(p;L))> =
    fold[large_category{i},ListF{i}(T),InitialAlgebra(ListF(T))](g) )
```

Fig. 24. Theorem `list_and_2_map_is_fold`

We can unfold its extract (and the extracts of other lemmata that were used in its proof) to obtain the actual function g with $and(map(p;\cdot)) = \mathsf{fold}\, g$. This function (with a few simplifications made by hand) is shown in Figure 25. The first and second component of the triple are the function's domain and codomain, respectively. The `if-then-else` statement is used to determine whether $x \in \mathbf{1} + (T \times \mathbb{B})$ is in the image of $\mathcal{L}_T(and(map(p;\cdot)))$. Three cases need to be distinguished: $x = inl\ \cdot$, $x = inr\ (y1, \text{true})$, and $x = inr\ (y1, \text{false})$. The latter can only occur if $p(t) = \text{false}$ for some $t \in T$; whether such a t exists is determined by the value of ϕ. If x is in the image of $\mathcal{L}_T(and(map(p;\cdot)))$, the `then` part is used to apply $and(map(p;\cdot)) \cdot (nil + cons)$ to an element $z \in \mathbf{1} + (T \times List(T))$ with $(\mathcal{L}_T(and(map(p;\cdot))))(z) = x$. Otherwise, an arbitrary boolean (in this case true) is returned in the `else` part.

```
< (ListF{i}(T)_0 𝔹)
, 𝔹
, λx.if case x
    of inl(_) => true
    | inr(<y1,y2>) => case y2
           of inl(_) => true
           | inr(_) => case φ
               of inl(_) => true
               | inr(_) => false
  then (<T List, 𝔹, λL.∧b(map(p;L))> ListF{i}(T)_dom_op
    InitialAlgebra(ListF(T))_arr).2.2
    (case x
    of inl(_) => <inl ·, Ax>
    | inr(<y1,y2>) => case y2
           of inl(_) => <inr <y1, []> , Ax>
           | inr(_) => case φ
               of inl(<t,_>) => <inr <y1, t::[]>, Ax>
               | inr(_) => arbitrary)
  else true
fi >
```

Fig. 25. A function g with $and(map(p;\cdot)) = \mathsf{fold}\, g$

The function shown in Figure 25 is unlikely to be more efficient than the initial composition of `and` and `map`, due to the increased overhead associated with each list element. However, we could further simplify the function by using $and(map(p;[])) = \text{true}$ and $and(map(p;t :: [])) = \text{false}$ and combining the two outermost `case` constructs (which have identical structure).

6.3 Constructively characterizing unfold

Now we consider reformulating the ($\Leftarrow$) direction of Theorem 6. To prove it, we identify additional assumptions under which the inclusion of images constructively implies the existence of prefactors. Dualizing our results for kernels and postfactors, one could suspect that (among other things) we need to be able to decide whether the domain of h is empty. However, it turns out that the classical proof of the ($\Leftarrow$) direction of Theorem 6 given in [6] can be simplified significantly. In particular, the dual of Lemma 2, although easily provable in Nuprl (see Fig. 26), turns out not to be needed.

```
∀F:Functor{i'}(large_category{i},large_category{i}).
 ∀T:{T:Coalgebra(F)| coalgebra_category(F)-terminal(T)} .
  ∀A:large_category{i}_Obj. Dec(A) ⇒
   Mor[large_category{i}](A,T_obj) ⇒ Mor[large_category{i}](A,F_O A)
```

Fig. 26. Theorem: `cohom_fun_implies_coalgebra_fun`

Therefore it is sufficient to replace the precondition $\mathsf{img}(out \cdot h) \subseteq \mathsf{img}(Fh)$ by the (classically equivalent) condition $\forall c \in \mathsf{img}(out \cdot h).\ \exists b \in FA. \quad c = (Fh)(b)$ to give a constructive proof. We first prove that the latter condition implies the existence of prefactors.

Lemma 5. *Suppose that* $f : B \to C$ *and* $h : A \to C$*, where* A*,* B*,* C *are sets. Then* $(\exists g : A \to B. \quad h = f \cdot g) \Longleftarrow (\forall c \in \mathsf{img}\, h.\ \exists b \in B. \quad c = f(b))$.

Proof. Assume $\forall c \in \mathsf{img}\, h.\ \exists b \in B. \quad c = f(b)$. Let $choice : \mathsf{img}\, h \to B$ be a function with $f(choice(c)) = c$ for all $c \in \mathsf{img}\, h$. Now define $g = choice \cdot h$. Then $(f \cdot g)(a) = f(choice(h(a))) = h(a)$ for every $a \in A$, hence $h = f \cdot g$. $\square$

Note the difference between Lemma 5 and [6, Lemma 5.3]: our proof does not need $A \to B \neq \emptyset$ as an additional assumption. Figure 27 shows the corresponding Nuprl theorem, which is proved in 19 steps. As usual, we prove a lifted version for arrows in the category of types. This lifted version is shown in Fig. 28; using the `image_inv_fun_implies_prefactor` lemma, it is proved in 45 steps.

```
∀A,B,C:𝕌ᵢ. ∀f:B → C. ∀h:A → C. (∃g:A → B. h = f o g)
 ⇐ (∀c:img[A,C] h. ∃b:B. c = f b)
```

Fig. 27. Theorem: `image_inv_fun_implies_prefactor`

```
∀A,B,C:large_category{i}_Obj. ∀f:Mor[large_category{i}](B,C).
 ∀h:Mor[large_category{i}](A,C).
  (∃g:Mor[large_category{i}](A,B). h = f large_category{i}_op g)
    ⇐ (∀c:img[A,C] h.2.2 ∃b:B. c = f.2.2 b)
```

Fig. 28. Theorem: `image_inv_fun_implies_prefactor_cat`

Our main result for anamorphisms is now immediate.

Theorem 9. *Suppose* $F : \mathcal{SET} \to \mathcal{SET}$ *is a functor with a terminal coalgebra* $\langle \nu F, out \rangle$*,* A *is a set, and* $h : A \to \nu F$*. Then*

$$(\exists g : A \to FA. \quad h = \mathit{unfold}\, g) \Longleftarrow (\forall c \in \mathit{img}(out \cdot h).\ \exists b \in FA. \quad c = (Fh)(b)).$$

Proof.

$$
\begin{array}{ll}
& \forall c \in \mathsf{img}(out \cdot h).\ \exists b \in FA. \quad c = (Fh)(b) \\
\Longrightarrow & \quad \{ \text{ Lemma 5 } \} \\
& \exists g : A \to FA. \quad out \cdot h = Fh \cdot g \\
\Longleftrightarrow & \quad \{ \text{ universal property } \} \\
& \exists g : A \to FA. \quad h = \mathsf{unfold}\, g.
\end{array}
$$

□

Theorem `image_inv_fun_implies_unfold`, shown in Fig. 29, is the corresponding Nuprl theorem. Again mostly due to well-formedness goals, the formal proof requires about 91 steps.

```
∀F:Functor{i'}(large_category{i},large_category{i}).
 ∀T:{T:Coalgebra(F)| coalgebra_category(F)-terminal(T)} .
  ∀A:large_category{i}_Obj. ∀h:Mor[large_category{i}](A,T_obj).
   (∃g:Coalgebra(F). h=unfold[large_category{i},F,T](g))
     ⇐ (∀c:img[A,F_O T_obj] (T_arr F_dom_op h).2.2
          ∃b:F_O A. c = (F_M h).2.2 b)
```

Fig. 29. Theorem: `image_inv_fun_implies_unfold`

7 Conclusions

We have presented a constructive characterization of fold and unfold which we believe is of interest, independent of the formalizations presented here. However, we *have* completely formalized these results in Nuprl. The extract of Thm. 7 was applied to a small example involving the reformulation of the program `all p L = and (map p L)` as a fold. The hardest part of that proof was to show that the inductive type $List(T)$ is in fact the object of an initial algebra. However, proofs of initiality or finality only need be done once for each data-type. We have also proven finality for the coinductive type $Stream(T)$ and exercised the extract of Thm. 9 on a simple stream-generating function.

The presented program transformations could be used in an optimizing compiler to transform any function that meets certain (rather simple) semantic criteria into a fold or unfold. No knowledge of the function's implementation is required. Of course this generality comes at a price: the semantic properties that must be verified are, like all non-trivial semantic properties, not decidable in general. The compiler could analyze the function in question to try and prove these properties automatically, it could rely on human guidance, or it could use a combination of both approaches.

In the longer term, we hope to incorporate a wide variety of program transformations into the framework outlined here.

References

1. Stuart Allen. *NuprlPrimitives - Explanations of Nuprl Primitives and Conventions.* Department of Computer Science, Cornell University, Ithaca, NY, 2003. `http://www.cs.cornell.edu/Info/People/sfa/Nuprl/NuprlPrimitives/`.
2. Richard S. Bird. Functional algorithm design. In Bernhard Moller, editor, *Mathematics of Program Construction '95*, volume 947 of *Lecture Notes in Computer Science*, pages 2–17. Springer-Verlag, 1995.
3. Rod M. Burstall and John Darlington. A transformation system for developing recursive programs. *Journal of the ACM*, 24(1):44–67, January 1977.
4. James Caldwell. Extracting recursion operators in Nuprl's type-theory. In A. Pettorossi, editor, *Eleventh International Workshop on Logic-based Program Synthesis, LOPSTR-02*, volume 2372 of *LNCS*, pages 124–131. Springer, 2002.
5. Robert L. Constable, Stuart F. Allen, H. M. Bromley, W. R. Cleaveland, J. F. Cremer, R. W. Harper, Douglas J. Howe, T. B. Knoblock, N. P. Mendler, P. Panangaden, James T. Sasaki, and Scott F. Smith. *Implementing Mathematics with the Nuprl Proof Development System.* Prentice-Hall, NJ, 1986.
6. Jeremy Gibbons, Graham Hutton, and Thorsten Altenkirch. When is a function a fold or an unfold? In Andrea Corradini, Marina Lenisa, and Ugo Montanari, editors, *Proceedings 4th Workshop on Coalgebraic Methods in Computer Science, CMCS'01, Genova, Italy, 6–7 Apr. 2001*, volume 44(1). Elsevier, Amsterdam, 2001.
7. Jeremy Gibbons and Geraint Jones. The under-appreciated unfold. In *Proceedings 3rd ACM SIGPLAN Int. Conf. on Functional Programming, ICFP'98, Baltimore, MD, USA, 26–29 Sept. 1998*, volume 34(1), pages 273–279. ACM Press, New York, 1998.
8. Graham Hutton. Fold and unfold for program semantics. In *Proceedings 3rd ACM SIGPLAN Int. Conf. on Functional Programming, ICFP'98, Baltimore, MD, USA, 26–29 Sept. 1998*, volume 34(1), pages 280–288. ACM Press, New York, 1998.
9. Graham Hutton. A tutorial on the universality and expressiveness of fold. *Journal of Functional Programming*, 9(4):355–372, 1999.
10. Saunders MacLane. *Categories for the Working Mathematician*, volume 5 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, 2nd edition, 1997. (1st ed., 1971).
11. G. Malcolm. Algebraic data types and program transformation. *Science of Computer Programming*, 14(2–3):255–280, 1990.
12. Per Martin-Löf. Constructive mathematics and computer programming. In *Sixth International Congress for Logic, Methodology, and Philosophy of Science*, pages 153–175, 1982.
13. Erik Meijer, Maarten Fokkinga, and Ross Paterson. Functional programming with bananas, lenses, envelopes and barbed wire. In J. Hughes, editor, *Proceedings 5th ACM Conf. on Functional Programming Languages and Computer Architecture, FPCA'91, Cambridge, MA, USA, 26–30 Aug 1991*, volume 523, pages 124–144. Springer-Verlag, Berlin, 1991.
14. P. Wadler. Deforestation: Transforming programs to eliminate trees. In *ESOP '88. European Symposium on Programming, Nancy, France, 1988*, volume 300 of *Lecture Notes in Computer Science*, pages 344–358. Berlin: Springer-Verlag, 1988.
15. Tjark Weber. Program transformations in Nuprl. Master's thesis, University of Wyoming, Laramie, WY, August 2002.