

Constructively Characterizing Fold and Unfold^{*}

Tjark Weber¹ and James Caldwell²

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 $h = \text{fold } g$ or $h = \text{unfold } g$? 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 $h = \text{fold } g$ or, as the case may be, $h = \text{unfold } g$.

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 $h = \text{fold } g$ or $h = \text{unfold } g$? The proofs have been formalized in Nuprl’s constructive type theory [1, 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 $h = \text{fold } g$ or, as the case may be, $h = \text{unfold } g$.

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 [16] 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 `fold` g where g is necessarily incomputable. We extend those results here and 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.

^{*} This work was supported by NSF grant CCR-9985239 and a DoD Multidisciplinary University Research Initiative (MURI) program administered by the Office of Naval Research under grant N00014-01-1-0765.

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 [16]. 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]. We remark that Nuprl is an LCF style tactic-based prover and so proofs are constructed by user interaction with the prover.

The formal Nuprl definition of the type *category* (up through type universe level i) is shown in Fig.1.¹ 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} def
  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`.

¹ Like the related type theory of Martin-Löf [12], Nuprl's type theory is predicative, supporting an unbounded cumulative hierarchy of type universes. Every universe is itself a type and every type is an element of some universe. U_i denotes the universe at level i where i is a polymorphic variable denoting any universe level.

The analog of the large category of sets in our type theoretic formulation is the category of types whose type 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 $f \in A \rightarrow B$ and $A, B \in \mathbb{U}_i$. The large category is defined in Fig.2.

```

large_category{i}  $\stackrel{\text{def}}{=}$ 
<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: $\text{large_category}\{i\} \in \text{Cat}\{i'\}$ *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)  $\stackrel{\text{def}}{=} C\_cod\ f = C\_dom\ g$ 
Mor[C](p,q)  $\stackrel{\text{def}}{=} \{f:C\_Arr \mid C\_dom\ f = p \wedge C\_cod\ f = q\}$ 
C-initial(p)  $\stackrel{\text{def}}{=} \forall q:C\_Obj. \exists !f:C\_Arr. C\_dom\ f = p \wedge C\_cod\ f = q$ 
C-terminal(p)  $\stackrel{\text{def}}{=} \forall q:C\_Obj. \exists !f:C\_Arr. C\_dom\ f = q \wedge C\_cod\ f = p$ 

```

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)  $\stackrel{\text{def}}{=}$ 
{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  $\stackrel{\text{def}}{=} F.1$ 
F_cod  $\stackrel{\text{def}}{=} F.2.1$ 

```

Fig. 4. Abstractions: functor, functor_dom and functor_cod

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

```

Algebra(F)  $\stackrel{\text{def}}{=}$ 
  p:F_dom_Obj  $\times$  {f:F_dom_Arr | F_dom_dom f = F_0 p  $\wedge$  F_dom_cod f = p}
Hom(F)  $\stackrel{\text{def}}{=}$ 
  A:Algebra(F)
   $\times$  B:Algebra(F)
   $\times$  {f:F_dom_Arr | (F_dom_dom f = A_obj)  $\wedge$  (F_dom_cod f = B_obj)
       $\wedge$  (f F_dom_op A_arr = B_arr F_dom_op (F_M f))}
algebra_category(F)  $\stackrel{\text{def}}{=}$  <Algebra(F), Hom(F),  $\lambda h.h\_dom$ ,  $\lambda h.h\_cod$ ,
   $\lambda h.g.h$  o_hom[F] g,  $\lambda A.id\_hom[F](A)$ >

```

Fig. 5. Abstractions: algebra, homomorphism and algebra_category

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

```

Coalgebra(F)  $\stackrel{\text{def}}{=}$ 
  p:F_dom_Obj  $\times$  {f:F_dom_Arr | F_dom_dom f = p  $\wedge$  F_dom_cod f = F_0 p}
Cohom(F)  $\stackrel{\text{def}}{=}$ 
  A:Coalgebra(F)
   $\times$  B:Coalgebra(F)
   $\times$  {f:F_dom_Arr | (F_dom_dom f = A_obj)  $\wedge$  (F_dom_cod f = B_obj)
       $\wedge$  ((F_M f) F_dom_op A_arr = B_arr F_dom_op f)}
coalgebra_category(F)  $\stackrel{\text{def}}{=}$  <Coalgebra(F), Cohom(F),  $\lambda h.h\_dom$ ,  $\lambda h.h\_cod$ ,
   $\lambda h.g.h$  o_cohom[F] g,  $\lambda A.id\_cohom[F](A)$ >

```

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 [15, 2, 8, 7, 9, 16].

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 $h : \langle \mu F, in \rangle \rightarrow \langle A, f \rangle$. 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 $h : \langle A, f \rangle \rightarrow \langle \nu F, out \rangle$.

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

$$\begin{array}{ccc}
 F(\mu F) & \xrightarrow{F(\text{fold } f)} & FA \\
 \downarrow in & & \downarrow f \\
 \mu F & \xrightarrow{\text{fold } f} & A
 \end{array}$$

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

We say an arrow h is a *catamorphism* if and only if it can be written as **fold** f for some arrow f .

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

$$\begin{array}{ccc}
 A & \xrightarrow{\text{unfold } f} & \nu F \\
 \downarrow f & & \downarrow out \\
 FA & \xrightarrow{F(\text{unfold } f)} & F(\nu F)
 \end{array}$$

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

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

Theorem 3 (Universal Properties for fold and unfold). Let \mathcal{C} be a category, $F : \mathcal{C} \rightarrow \mathcal{C}$ a functor and $\langle \mu F, in \rangle$ an initial algebra (or in the case of **unfold**, let $\langle \nu F, out \rangle$ be a terminal coalgebra). Furthermore, suppose that $\langle A, f \rangle$ is an algebra (or a coalgebra) and that $h : \mu F \rightarrow A$ (or, in the case of **unfold**, $h : A \rightarrow \nu F$). Then

$$\begin{aligned}
 h = \text{fold } f &\iff h \cdot in = f \cdot Fh \\
 h = \text{unfold } f &\iff Fh \cdot f = out \cdot h.
 \end{aligned}$$

Based on their universal properties, formalize `fold` and `unfold` as relations (shown in Fig.9). The well-formedness theorems state that `fold` and `unfold` inhabit \mathbb{P} . We remark that g is unique when $g = \text{unfold } f$ or $g = \text{fold } f$.

$\begin{aligned} \text{h=fold}[C,F,I](f) &\stackrel{\text{def}}{=} (\text{F_dom_dom } h = \text{I_obj}) \text{ c} \wedge (\text{F_dom_cod } h = \text{f_obj}) \\ &\quad \text{c} \wedge (h \text{ F_dom_op } \text{I_arr} = \text{f_arr } \text{F_dom_op } (\text{F_M } h)) \\ \text{h=unfold}[C,F,T](f) &\stackrel{\text{def}}{=} (\text{F_dom_dom } h = \text{f_obj}) \text{ c} \wedge (\text{F_dom_cod } h = \text{T_obj}) \\ &\quad \text{c} \wedge ((\text{F_M } h) \text{ F_dom_op } \text{f_arr} = \text{T_arr } \text{F_dom_op } h) \end{aligned}$
--

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 $h : \mu F \rightarrow A$ is a catamorphism if and only if $h \cdot \text{in} = f \cdot Fh$ for some arrow $f : FA \rightarrow A$. 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 $h : A \rightarrow \nu F$ is an anamorphism if and only if $Fh \cdot f = \text{out} \cdot h$ for some arrow $f : A \rightarrow FA$. Again, to show that f exists or methods to construct it are not given.

E. Meijer, M. Fokkinga, and R. Paterson [14] give the following results regarding left and right invertible arrows.

Definition 4 (Left and Right Invertible) *Let \mathcal{C} be a category and f an arrow in \mathcal{C} .*

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

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

$\begin{aligned} \text{left-invertible}[C](f) &\stackrel{\text{def}}{=} \\ &\quad \exists g:\{g:C_Arr \mid C\text{-composable}(f,g)\} . g \text{ C_op } f = \text{C_id } (C_dom \ f) \\ \text{right-invertible}[C](f) &\stackrel{\text{def}}{=} \\ &\quad \exists g:\{g:C_Arr \mid C\text{-composable}(g,f)\} . f \text{ C_op } g = \text{C_id } (C_cod \ f) \end{aligned}$

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

The following theorems provide tools to show when f exists.

Theorem 5. *If \mathcal{C} is a category, $F : \mathcal{C} \rightarrow \mathcal{C}$ is a functor with an initial algebra $\langle \mu F, \text{in} \rangle$, and $h : \mu F \rightarrow A$ is a left-invertible arrow in \mathcal{C} , then, for some arrow $f : FA \rightarrow A$, $h = \text{fold } f$.*

Theorem 6. *If \mathcal{C} is a category, $F : \mathcal{C} \rightarrow \mathcal{C}$ is a functor with a terminal coalgebra $\langle \nu F, out \rangle$, and $h : A \rightarrow \nu F$ is a right-invertible arrow in \mathcal{C} , then, for some arrow $f : A \rightarrow FA$, $h = unfold\ f$.*

The Nuprl theorems formalizing these results are shown in Fig.11; they 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 extract² 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 $\langle F_dom_dom\ h, (F_M\ g)\ F_dom_op\ (T_arr\ F_dom_op\ h) \rangle$. 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 7 (Gibbons, Hutton, Altenkirch: fold). *Let $F : \mathcal{SET} \rightarrow \mathcal{SET}$ be a functor with an initial algebra $\langle \mu F, in \rangle$, A be a set, and $h : \mu F \rightarrow A$. Then $(\exists g : FA \rightarrow A. h = fold\ g) \iff ker(Fh) \subseteq ker(h \cdot in)$.*

Here, $ker\ f$, the *kernel* of a function $f : A \rightarrow B$, 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 the classical theorem is too strong in the following sense: there exists a function h such that h is computable and such that $ker(Fh) \subseteq ker(h \cdot in)$ but where g , which exists by Thm.7, is necessarily incomputable [16].

The following theorem characterizes the dual unfold.

² The extract has been simplified by unfolding definitions, performing β -reductions and α -renaming selected variables to make the code more readable.

Theorem 8 (Gibbons, Hutton, Altenkirch: unfold). *Let $F : \mathcal{SET} \rightarrow \mathcal{SET}$ be a functor with a terminal coalgebra $\langle \nu F, out \rangle$, A be a set, and $h : A \rightarrow \nu F$. Then $(\exists g : A \rightarrow FA. h = unfold\ g) \iff img(out \cdot h) \subseteq img(Fh)$.*

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

$\ker[A,B]\ f \stackrel{\text{def}}{=} \{aa:A \times A \mid f\ aa.1 = f\ aa.2\}$ $img[A,B]\ f \stackrel{\text{def}}{=} \{b:B \mid \exists a:A. b = f\ a\}$
--

Fig. 13. Abstractions: kernel and image

6 A Constructive Characterization of fold and unfold

Translating the statements of Thms. 7 and 8 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 [16]. 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.7 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 $\{x : \mathbb{Z} \mid x < x\}$ 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.

We address the empty type issue by adding an antecedent of the form $Dec\{T\}$ for those types whose emptiness must be decided³. Computationally, a program extracted from a theorem with this antecedent expects one of its inputs to be a witness for the inhabitation of T or a witness that T is uninhabited; for an inhabited type the witness is typically trivial, *i.e.* it is a term of the form $inl(a)$ where a is any inhabitant of T .

Here is the statement of the refined theorem (still in terms of the category \mathcal{SET}) corresponding to the (\Leftarrow) part of Thm.7.

³ Note that $Dec\{T\} \stackrel{\text{def}}{=} T \vee \neg T$

Theorem 9. *Let $F : \mathcal{SET} \rightarrow \mathcal{SET}$ 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 $h : \mu F \rightarrow A$. Furthermore, suppose that for every $b \in FA$ we can decide whether $b = (Fh)(a)$ for some $a \in F(\mu F)$. Then $(\exists g : FA \rightarrow A. h = \text{fold } g) \iff \ker(Fh) \subseteq \ker(h \cdot in)$.*

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

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_0 A. Dec(∃a:F_0 I_obj. b = (F_M h).2.2 a)) ⇒
      ((∃g:Algebra(F). h=fold[large_category{i},F,I](g) )
        ⇐ ker[F_0 I_obj,large_category{i}_cod (F_M h)] (F_M h).2.2
          ⊆ ker[F_0 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 $f : A \rightarrow B$ and $h : A \rightarrow C$. 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 $(\exists g : B \rightarrow C. h = g \cdot f) \iff (\ker f \subseteq \ker h \wedge B \rightarrow C \neq \emptyset)$.*

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

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

If $C \neq \emptyset$, let c be an arbitrary element in C . Let $choice : \{b \in B \mid \exists a \in A. b = f(a)\} \rightarrow A$ be a function with $f(choice(b)) = b$ for all $b \in B$.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 $h = g \cdot f$. \square

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

⁴ To prove that the function $choice$ exists, we use the so-called *axiom of choice* which is provable in constructive type theory [13] and is a theorem in the Nuprl standard library.

$$\begin{array}{l}
\forall A, B, C: \mathbb{U}_i. \quad \forall f: A \rightarrow B. \quad \forall h: A \rightarrow C. \\
\text{Dec}(C) \Rightarrow (\forall b: B. \text{Dec}(\exists a: A. b = f a)) \Rightarrow \\
((\exists g: B \rightarrow C. h = g \circ f) \Leftarrow \ker[A, B]f \subseteq \ker[A, C]h \wedge B \rightarrow C)
\end{array}$$

Fig. 15. Theorem `kernel_inclusion_implies_postfactor`

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 sets. Despite the use of the original lemma in the proof of the lifted version, the proof is about 71 steps long.

The second lemma required for the proof of Thm.9 is stated below.

$$\begin{array}{l}
\forall A, B, C: \text{large_category}\{i\}_Obj. \quad \forall f: \text{Mor}[\text{large_category}\{i\}](A, B). \\
\forall h: \text{Mor}[\text{large_category}\{i\}](A, C). \\
\text{Dec}(C) \Rightarrow (\forall b: B. \text{Dec}(\exists a: A. b = f.2.2 a)) \Rightarrow \\
((\exists g: \text{Mor}[\text{large_category}\{i\}](B, C). h = g \text{ large_category}\{i\}_op f) \\
\Leftarrow \ker[A, B]f.2.2 \subseteq \ker[A, C]h.2.2 \wedge \text{Mor}[\text{large_category}\{i\}](B, C))
\end{array}$$

Fig. 16. Theorem `kernel_inclusion_implies_postfactor_cat`

Lemma 2. *If $F : \mathcal{SET} \rightarrow \mathcal{SET}$ 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*

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

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

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

Therefore $FA \rightarrow A \neq \emptyset$ in either case. \square

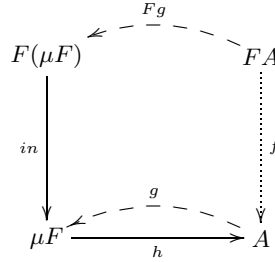


Fig. 17. $\mu F \rightarrow A \neq \emptyset \implies FA \rightarrow A \neq \emptyset$.

Figure 17 illustrates the situation: Given a function $h : \mu F \rightarrow A$, we can find a function $f : FA \rightarrow A$. The functions $g : A \rightarrow \mu F$ and $Fg : FA \rightarrow F(\mu F)$ are needed only in the case $A = \emptyset$. If $A \neq \emptyset$, they may not exist—but we can construct a function $f : FA \rightarrow A$ 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_0 A,A)

```

Fig. 18. Theorem `hom_fun_implies_algebra_fun`

We are now ready to prove Theorem 9.

Proof.

$$\begin{aligned}
& \ker(Fh) \subseteq \ker(h \cdot in) \\
\iff & \{ \text{Lemma 2, } h : \mu F \rightarrow A \} \\
& \ker(Fh) \subseteq \ker(h \cdot in) \wedge FA \rightarrow A \neq \emptyset \\
\implies & \{ \text{Lemma 1} \} \\
& \exists g : FA \rightarrow A. \quad h \cdot in = g \cdot Fh \\
\iff & \{ \text{universal property} \} \\
& \exists g : FA \rightarrow A. \quad h = \text{fold } g.
\end{aligned}$$

□

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 9. Closer inspection of the proof of Theorem 9 however shows that when h is surjective, we do not need the additional assumption that we can decide whether A is empty.

Theorem 10. *Suppose $F : \mathcal{SET} \rightarrow \mathcal{SET}$ is a functor with an initial algebra $(\mu F, in)$, and $h : \mu F \rightarrow A$ is surjective. Then*

$$(\exists g : FA \rightarrow A. \quad h = \text{fold } g) \iff \ker(Fh) \subseteq \ker(h \cdot in).$$

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

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

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

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 5.

Proof. For the (\Rightarrow) direction, suppose f is surjective. Then there exists a function $g : B \rightarrow A$ 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 $g : B \rightarrow A$. Now let $b \in B$. Then $f(g(b)) = (f \cdot g)(b) = (id(B))(b) = b$. Therefore f is surjective. \square

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.

$$\forall A, B: \mathbb{U}_i. \forall f: A \rightarrow B. \\ \text{Surj}(A; B; f) \iff \text{right-invertible}[\text{large_category}\{i\}](\langle A, B, f \rangle)$$

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`.

$$\forall f: \text{large_category}\{i\}\text{-Arr} \\ \text{Surj}(\text{large_category}\{i\}\text{-dom } f; \text{large_category}\{i\}\text{-cod } f; f.2.2) \\ \iff \text{right-invertible}[\text{large_category}\{i\}](f)$$

Fig. 20. Theorem `surjective_iff_right_invertible_cat`

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

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

$$(\exists g : B \rightarrow C. \ h = g \cdot f) \iff \ker f \subseteq \ker h.$$

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

Let $choice : B \rightarrow A$ 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 $g : B \rightarrow C$ 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 $g(f(a)) = h(choice(f(a))) = h(a)$. \square

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.

$$\forall A, B, C: \mathbb{U}_i. \forall f: A \rightarrow B. \forall h: A \rightarrow C. \text{Surj}(A; B; f) \Rightarrow \\ ((\exists g: B \rightarrow C. h = g \circ f) \iff \ker[A, B] f \subseteq \ker[A, C] h)$$

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.

$$\begin{aligned} & \forall A, B, C : \text{large_category}\{i\}\text{-Obj}. \quad \forall f : \text{Mor}[\text{large_category}\{i\}](A, B). \\ & \forall h : \text{Mor}[\text{large_category}\{i\}](A, C). \quad \text{Surj}(A; B; f.2.2) \Rightarrow \\ & \quad ((\exists g : \text{Mor}[\text{large_category}\{i\}](B, C). \quad h = g \text{ large_category}\{i\}\text{-op } f) \\ & \quad \Leftarrow \text{ker}[A, B] \text{ f.2.2} \subseteq \text{ker}[A, C] \text{ h.2.2}) \end{aligned}$$

Fig. 22. Theorem `kernel_inclusion_implies_postfactor_surjective_cat`

Using the two Lemmata 3 and 4, we can now prove Theorem 10.

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

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

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

$$\begin{aligned} & \text{ker}(Fh) \subseteq \text{ker}(h \cdot in) \\ \Rightarrow & \quad \{ \text{Lemma 4} \} \\ & \quad \exists g : FA \rightarrow A. \quad h \cdot in = g \cdot Fh \\ \Leftrightarrow & \quad \{ \text{universal property} \} \\ & \quad \exists g : FA \rightarrow A. \quad h = \text{fold } g \end{aligned}$$

completing the proof. \square

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 Constructively characterizing unfold

Now we consider reformulating the (\Leftarrow) direction of Theorem 8. 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

$$\begin{aligned}
& \forall F:\text{Functor}\{i'\}(\text{large_category}\{i\}, \text{large_category}\{i\}). \\
& \forall I:\{I:\text{Algebra}(F) \mid \text{algebra_category}(F)\text{-initial}(I)\} . \\
& \forall A:\text{large_category}\{i\}\text{-Obj}. \forall h:\text{Mor}[\text{large_category}\{i\}](I_{\text{obj}}, A). \\
& \text{Surj}(I_{\text{obj}}; A; h.2.2) \Rightarrow \\
& \quad ((\exists g:\text{Algebra}(F). h = \text{fold}[\text{large_category}\{i\}, F, I](g)) \\
& \quad \Leftarrow \ker[F_0 I_{\text{obj}}, \text{large_category}\{i\}\text{-cod}(F_M h)](F_M h).2.2 \\
& \quad \quad \subseteq \ker[F_0 I_{\text{obj}}, \text{large_category}\{i\}\text{-cod}(h F_{\text{dom_op}} I_{\text{arr}})] \\
& \quad \quad (h F_{\text{dom_op}} I_{\text{arr}}).2.2)
\end{aligned}$$

Fig. 23. Theorem `kernel_inclusion_implies_fold_surjective`

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 8 given in [6] can be simplified significantly. In particular, the dual of Lemma 2, although easily provable in Nuprl (see Fig.24), turns out not to be needed.

$$\begin{aligned}
& \forall F:\text{Functor}\{i'\}(\text{large_category}\{i\}, \text{large_category}\{i\}). \\
& \forall T:\{T:\text{Coalgebra}(F) \mid \text{coalgebra_category}(F)\text{-terminal}(T)\} . \\
& \forall A:\text{large_category}\{i\}\text{-Obj}. \text{Dec}(A) \Rightarrow \\
& \quad \text{Mor}[\text{large_category}\{i\}](A, T_{\text{obj}}) \Rightarrow \text{Mor}[\text{large_category}\{i\}](A, F_0 A)
\end{aligned}$$

Fig. 24. Theorem `cohom_fun_implies_coalgebra_fun`

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

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

Proof. Assume $\forall c \in \text{img} h. \exists b \in B. c = f(b)$. Let $\text{choice} : \text{img} h \rightarrow B$ be a function with $f(\text{choice}(c)) = c$ for all $c \in \text{img} h$. Now define $g = \text{choice} \cdot h$. Then $(f \cdot g)(a) = f(\text{choice}(h(a))) = h(a)$ 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 $A \rightarrow B \neq \emptyset$ as an additional assumption. Figure 25 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.26; using the `image_inv_fun_implies_prefactor` lemma, it is proved in 45 steps.

$$\begin{aligned}
& \forall A, B, C:\mathbb{U}_i. \forall f:B \rightarrow C. \forall h:A \rightarrow C. (\exists g:A \rightarrow B. h = f \circ g) \\
& \Leftarrow (\forall c:\text{img}[A, C] h. \exists b:B. c = f b)
\end{aligned}$$

Fig. 25. Theorem `image_inv_fun_implies_prefactor`

Our main result for anamorphisms is now immediate.

Theorem 11. *Suppose $F : \mathcal{SET} \rightarrow \mathcal{SET}$ is a functor with a terminal coalgebra $\langle \nu F, \text{out} \rangle$, A is a set, and $h : A \rightarrow \nu F$. Then*

$$(\exists g : A \rightarrow FA. h = \text{unfold } g) \Leftarrow (\forall c \in \text{img}(\text{out} \cdot h). \exists b \in FA. c = (Fh)(b)).$$

$$\begin{aligned}
& \forall A, B, C: \text{large_category}\{i\}\text{-Obj}. \quad \forall f: \text{Mor}[\text{large_category}\{i\}](B, C). \\
& \quad \forall h: \text{Mor}[\text{large_category}\{i\}](A, C). \\
& \quad (\exists g: \text{Mor}[\text{large_category}\{i\}](A, B). \quad h = f \text{ large_category}\{i\}\text{-op } g) \\
& \quad \Leftarrow (\forall c: \text{img}[A, C] \text{ h.2.2 } \exists b: B. \quad c = f.2.2 \text{ b})
\end{aligned}$$

Fig. 26. Theorem `image_inv_fun_implies_prefactor_cat`

Proof.

$$\begin{aligned}
& \forall c \in \text{img}(out \cdot h). \quad \exists b \in FA. \quad c = (Fh)(b) \\
\implies & \quad \{ \text{Lemma 5} \} \\
& \exists g: A \rightarrow FA. \quad out \cdot h = Fh \cdot g \\
\iff & \quad \{ \text{universal property} \} \\
& \exists g: A \rightarrow FA. \quad h = \text{unfold } g.
\end{aligned}$$

□

Theorem `image_inv_fun_implies_unfold`, shown in Fig.27, is the corresponding Nuprl theorem. Again mostly due to well-formedness goals, the formal proof requires about 91 steps.

$$\begin{aligned}
& \forall F: \text{Functor}\{i'\}(\text{large_category}\{i\}, \text{large_category}\{i\}). \\
& \quad \forall T: \{T: \text{Coalgebra}(F) \mid \text{coalgebra_category}(F)\text{-terminal}(T)\} . \\
& \quad \forall A: \text{large_category}\{i\}\text{-Obj}. \quad \forall h: \text{Mor}[\text{large_category}\{i\}](A, T\text{-obj}). \\
& \quad (\exists g: \text{Coalgebra}(F). \quad h = \text{unfold}[\text{large_category}\{i\}, F, T](g)) \\
& \quad \Leftarrow (\forall c: \text{img}[A, F_0 \text{ T_obj}] \text{ (T_arr } F\text{-dom_op } h).2.2 \\
& \quad \quad \exists b: F_0 \text{ A}. \quad c = (F_M \text{ h}).2.2 \text{ b})
\end{aligned}$$

Fig. 27. 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. In [16] the extract of Thm.9 was applied to a simple 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 proved that the coinductive type `Stream(T)` is final and we plan to exercise Thm.27 in this data-type. In the longer term, we hope to incorporate a wide variety of program transformations into the framework outlined here.

8 Thanks

The authors would like to thank the anonymous referees for their careful reading and suggestions.

References

1. Stuart Allen. *Nuprl Basics*. Cornell University, 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. R. Burstall and J. Darlington. A transformation system for developing recursive programs. *J. Assoc. Comput. Mach.*, 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 153–75. Amsterdam:North Holland, 1982.
13. Per Martin-Löf. Constructive mathematics and computer programming. In *Sixth International Congress for Logic, Methodology, and Philosophy of Science*, pages 153–175, 1982.
14. 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.
15. 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.
16. Tjark Weber. *Program Transformations in Nuprl*. Master’s thesis, University of Wyoming, Laramie, WY, August 2002 <http://www4.in.tum.de/~webertj>.