Constructively Characterizing Fold and Unfold

Tjark Weber and James Caldwell

webertj@in.tum.de, jlc@uwyo.edu

Technische Universität München

University of Wyoming

Motivation

Do all elements in a list `xs` satisfy some predicate `p`?

- `all p xs = and (map p xs)`

Motivation

Do all elements in a list `xs` satisfy some predicate `p`?

- `all p xs = and (map p xs)`
- `all p xs = foldr (`λ`x,y.p x `$\wedge$` y) True xs`,

  where

  ```
  foldr f e [] = e,
  foldr f e (x:xs) = f x (foldr f e xs)
  ```

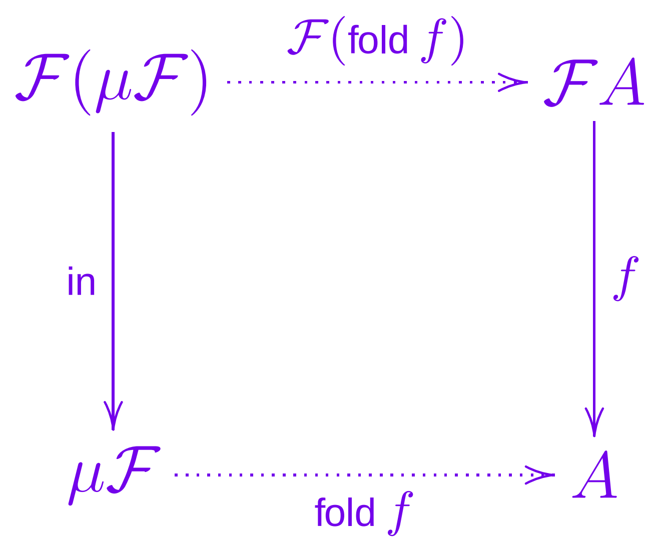
The second version is more efficient.

A little category theory

An *algebra* for a functor $\mathcal{F}$ is a pair (A, f) with

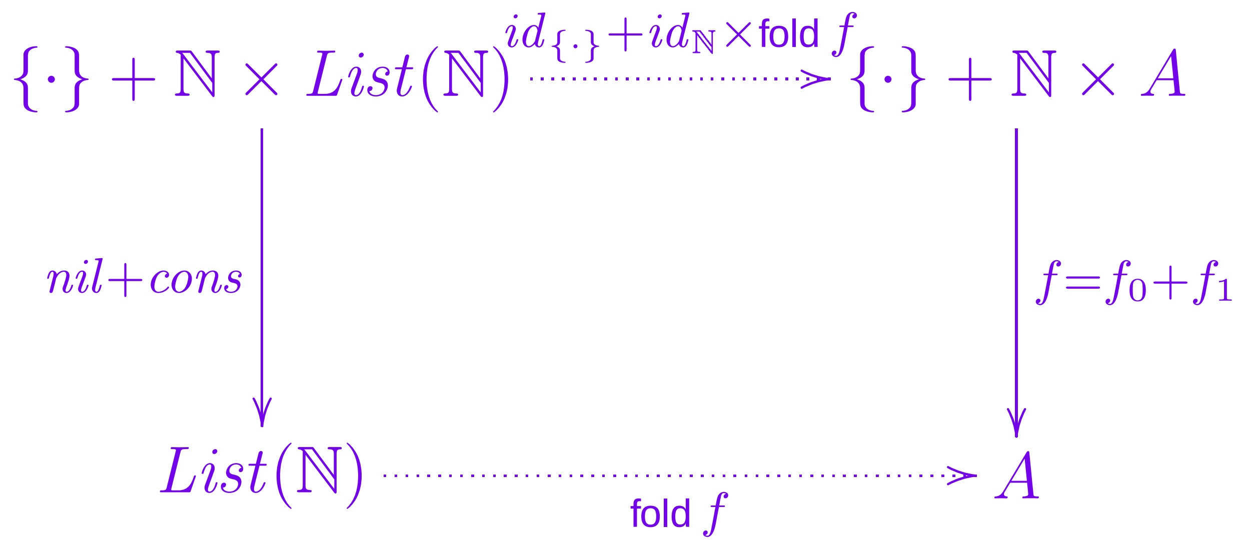
$$f : \mathcal{F}A \to A.$$

An *initial* algebra $(\mu\mathcal{F}, \mathsf{in})$ for a functor $\mathcal{F}$ has a unique homomorphism to any other such algebra:



Lists as initial algebra

For instance, with $\mathcal{F}X = \{\cdot\} + (\mathbb{N} \times X)$, an initial algebra is $\mu\mathcal{F}$ = (finite) lists of naturals, and in = nil + cons.



Examples of folds are sum, length, max, . . .

When is a function a fold?

Given a function h, when is $h = \mathsf{fold}\ g$ for some function g?

When is a function a fold?

Given a function h, when is $h = \mathsf{fold}\, g$ for some function g?

The *kernel* of a function $f : A \to B$ is the set

$$\ker f = \{(a, a') \in A \times A \mid f(a) = f(a')\}.$$

[GHA01]: Suppose $\mathcal{F} : SET \to SET$ is a functor with an initial algebra $(\mu\mathcal{F}, \mathsf{in})$, and $h : \mu\mathcal{F} \to A$. Then

$$\exists g : \mathcal{F}A \to A.\, h = \mathsf{fold}\, g \iff \ker \mathcal{F}h \subseteq \ker(h \cdot \mathsf{in}).$$

How to compute fold^{-1}?

Given a function h, when (and how) can we *compute* a function g such that $h = \text{fold}\, g$?

How to compute fold^{-1}?

Given a function h, when (and how) can we *compute* a function g such that $h = \mathsf{fold}\, g$?

$$\exists g : \mathcal{F}A \to A.\, h = \mathsf{fold}\, g \iff \ker \mathcal{F}h \subseteq \ker(h \cdot \mathsf{in})$$

- "$\Rightarrow$" is constructively valid.
- "$\Leftarrow$" however is *not*: There are computable functions h with $\ker \mathcal{F}h \subseteq \ker(h \cdot \mathsf{in})$ such that no *computable* function g satisfies $h = \mathsf{fold}\, g$.

Nuprl

- A Computational Type Theory (based on Martin-Löf 1980)
- An LCF style interactive tactic based prover
- Tools to extract “correct-by-construction” programs from formal proofs
- http://www.nuprl.org/

Abstraction category

```
* ABS category
Cat{i} ==
    Obj:𝕌ᵢ
    × Arr:𝕌ᵢ
    × dom:(Arr → Obj)
    × cod:(Arr → Obj)
    × o:{o:(g:Arr → f:{f:Arr| cod f = dom g} →
        {h:Arr| dom h = dom f ∧ cod h = cod g}) |
          ∀f,g,h:Arr. cod f = dom g ∧ cod g = dom h ⟹
            (h o g) o f = h o (g o f)}
    × {id:(p:Obj → {f:Arr| dom f = p ∧ cod f = p}) |
        ∀f:Arr. (id (cod f)) o f = f ∧
          f o (id (dom f)) = f}
```

A constructive result

Suppose $\mathcal{F} : TYP \to TYP$ is a functor with an initial algebra $(\mu\mathcal{F}, \mathsf{in})$, $h : \mu\mathcal{F} \to A$, we can decide whether A is empty, and for each $b \in \mathcal{F}A$ we can decide whether there exists some $a \in \mathcal{F}(\mu\mathcal{F})$ with $b = (\mathcal{F}h)(a)$. Then

$$\exists g : \mathcal{F}A \to A.\, h = \mathsf{fold}\, g \Longleftarrow \ker \mathcal{F}h \subseteq \ker(h \cdot \mathsf{in}).$$

A constructive result

Suppose $\mathcal{F} : TYP \rightarrow TYP$ is a functor with an initial algebra $(\mu\mathcal{F}, \mathsf{in})$, $h : \mu\mathcal{F} \rightarrow A$, we can decide whether A is empty, and for each $b \in \mathcal{F}A$ we can decide whether there exists some $a \in \mathcal{F}(\mu\mathcal{F})$ with $b = (\mathcal{F}h)(a)$. Then

$$\exists g : \mathcal{F}A \rightarrow A.\, h = \mathsf{fold}\, g \Longleftarrow \ker \mathcal{F}h \subseteq \ker(h \cdot \mathsf{in}).$$

- Replaced classical reasoning
- Sets as types: extensional vs. intensional equality
- Case splits justified by the additional premises

A result for right-invertible functions

Suppose $\mathcal{F} : TYP \to TYP$ is a functor with an initial algebra $(\mu\mathcal{F}, \mathsf{in})$, $h : \mu\mathcal{F} \to A$, we can decide whether A is empty, and for each $b \in \mathcal{F}A$ we can decide whether there exists some $a \in \mathcal{F}(\mu\mathcal{F})$ with $b = (\mathcal{F}h)(a)$. Then

$$\exists g : \mathcal{F}A \to A.\, h = \mathsf{fold}\, g \Longleftarrow \ker \mathcal{F}h \subseteq \ker(h \cdot \mathsf{in}).$$

A result for right-invertible functions

Suppose $\mathcal{F} : TYP \to TYP$ is a functor with an initial algebra $(\mu\mathcal{F}, \mathsf{in})$, $h : \mu\mathcal{F} \to A$, we can decide whether A is empty, and for each $b \in \mathcal{F}A$ we can decide whether there exists some $a \in \mathcal{F}(\mu\mathcal{F})$ with $b = (\mathcal{F}h)(a)$. Then

$$\exists g : \mathcal{F}A \to A.\, h = \mathsf{fold}\, g \Longleftarrow \ker \mathcal{F}h \subseteq \ker(h \cdot \mathsf{in}).$$

A result for right-invertible functions

Suppose $\mathcal{F} : TYP \to TYP$ is a functor with an initial algebra $(\mu\mathcal{F}, \mathsf{in})$, $h : \mu\mathcal{F} \to A$, and for each $b \in \mathcal{F}A$ there exists some $a \in \mathcal{F}(\mu\mathcal{F})$ with $b = (\mathcal{F}h)(a)$. Then

$$\exists g : \mathcal{F}A \to A.\, h = \mathsf{fold}\, g \Longleftarrow \ker \mathcal{F}h \subseteq \ker(h \cdot \mathsf{in}).$$

Examples

Embedded in the proofs are algorithms to compute g from h (accompanied by the evidence that h satisfies the required conditions).

- sum, length, max, … are right-invertible, and thus can be written as a fold.
- `all p` can be written as a fold if we can decide whether there exists an `x` with `p x = False`.

Transforming `all` into a fold

$g : \{\cdot\} + (\mathbb{N} \times \mathbb{B}) \to \mathbb{B}$

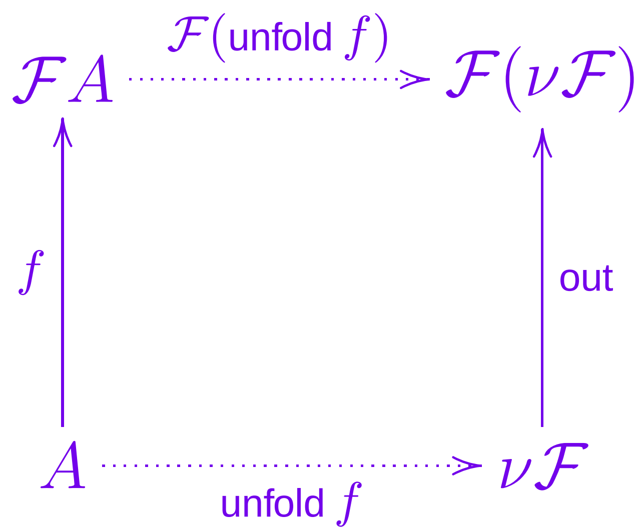
```
λx.if
   case x of inl _ => True
   | inr <_,b> => if b then True
       else case φ of inl _ => True
             | inr _ => False
then
   (λxs.and (map p xs)) o (nil+cons)
   (case x of inl _ => inl ·
    | inr <n,b> => if b then inr <n,[]>
        else case φ of inl <t,_> => inr <n,t:[]>
              | inr _ => arbitrary)
else
   True
```

unfold

A *coalgebra* for a functor $\mathcal{F}$ is a pair (A, f) with

$$f : A \rightarrow \mathcal{F}A.$$

A *terminal* coalgebra $(\nu\mathcal{F}, \mathsf{out})$ for a functor $\mathcal{F}$ has a unique cohomomorphism from any other such coalgebra:



Classical theorem for unfolds

[GHA01]: Suppose $\mathcal{F} : SET \to SET$ is a functor with a terminal coalgebra $(\nu\mathcal{F}, \mathsf{out})$, and $h : A \to \nu\mathcal{F}$. Then

$$\exists g : A \to \mathcal{F}A.\, h = \mathsf{unfold}\, g \iff \mathrm{img}(\mathsf{out} \cdot h) \subseteq \mathrm{img}\, \mathcal{F}h.$$

- Simply dual to the classical theorem for folds
- Again, "$\Rightarrow$" is constructively valid

Constructive theorem for unfolds

Suppose $\mathcal{F} : TYP \rightarrow TYP$ is a functor with a terminal coalgebra $(\nu\mathcal{F}, \mathsf{out})$, and $h : A \rightarrow \nu\mathcal{F}$. Then

$$\exists g : A \rightarrow \mathcal{F}A.\, h = \mathsf{unfold}\, g \Longleftarrow$$
$$\forall c \in \mathrm{img}(\mathsf{out} \cdot h).\, \exists b \in \mathcal{F}A.\, c = (\mathcal{F}h)(b).$$

- *Not* just dual to the constructive theorem for folds
- Very similar to the classical theorem for unfolds (but different proof)

Conclusions

- Constructive characterization of fold and unfold
- Simplification of the classical proofs
- Complete formalization in Nuprl
- Extraction of “correct-by-construction” program transformations from the proofs
- Other program transformations can be incorporated into the same framework