Quotients of Bounded Natural Functors

Basil Führer
Andreas Lochbihler
Joshua Schneider
Dmitriy Traytel
Dramatis personae

Andreas
Isabelle Expert

Dmitriy
Working Formalizer and Narrator

Isabelle
Proof Assistant

The characters and incidents portrayed and the names used herein are fictitious and any resemblance to the names, character, or history of any person is coincidental and unintentional.
A formalization problem

datatype \( a \text{ re} = \text{Atom } a \mid \text{Alt } (a \text{ re}) (a \text{ re}) \mid \text{Conc } (a \text{ re}) (a \text{ re}) \mid \text{Star } (a \text{ re}) \)
A formalization problem

```
datatype a re = Atom a | Alt (a re) (a re) | Conc (a re) (a re) | Star (a re)
```

```
datatype ldl = Prop string | And ldl ldl | Neg ldl | Match (ldl re)
```
A formalization problem

datatype \( a \, re = \text{Atom} \, a \mid \text{Alt} \, (a \, re) \, (a \, re) \mid \text{Conc} \, (a \, re) \, (a \, re) \mid \text{Star} \, (a \, re) \)
A formalization problem

datatype $a re = \text{Atom } a \mid \text{Alt } (a re) (a re) \mid \text{Conc } (a re) (a re) \mid \text{Star } (a re)$

inductive $\sim_{\text{ACI}} \text{ where}$

$\text{Alt } (\text{Alt } r s) t \sim_{\text{ACI}} \text{Alt } r (\text{Alt } s t)$
$\text{Alt } r s \sim_{\text{ACI}} \text{Alt } s r$
$\text{Alt } r r \sim_{\text{ACI}} r$
$\text{r } \sim_{\text{ACI}} \text{r’}$
$\text{s } \sim_{\text{ACI}} \text{s’}$
$\text{Alt } r s \sim_{\text{ACI}} \text{Alt } r' s'$
$\text{Alt } r s \sim_{\text{ACI}} \text{Alt } r' s'$

$\text{Conc } r s \sim_{\text{ACI}} \text{Conc } r' s'$
$\text{Conc } r s \sim_{\text{ACI}} \text{Conc } r' s'$

$\text{Star } r \sim_{\text{ACI}} \text{Star } r'$

$\text{r } \sim_{\text{ACI}} \text{r}$

$\text{r } \sim_{\text{ACI}} \text{r’}$

$\text{s } \sim_{\text{ACI}} \text{r}$

$\text{s } \sim_{\text{ACI}} \text{r’}$

$\text{t } \sim_{\text{ACI}} \text{r}$

quotient_type $a re_{\text{ACI}} = a re \sim_{\text{ACI}}$

datatype $ldl = \text{Prop } string \mid \text{And } ldl ldl \mid \text{Neg } ldl \mid \text{Match } (ldl re_{\text{ACI}})$
A formalization problem

**datatype** $a \text{ re} = \text{Atom } a \mid \text{Alt } (a \text{ re}) (a \text{ re}) \mid \text{Conc } (a \text{ re}) (a \text{ re}) \mid \text{Star } (a \text{ re})$

**inductive** $\sim_{\text{ACI}}$ where

<table>
<thead>
<tr>
<th>Rule</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$\text{Alt } r s \sim_{\text{ACI}} \text{Alt } s r$</td>
</tr>
<tr>
<td>2</td>
<td>$\text{Alt } r s \sim_{\text{ACI}} \text{Alt } s r$</td>
</tr>
<tr>
<td>3</td>
<td>$\text{Alt } r r \sim_{\text{ACI}} r$</td>
</tr>
<tr>
<td>4</td>
<td>$\text{Alt } r s \sim_{\text{ACI}} \text{Alt } r' s'$</td>
</tr>
<tr>
<td>5</td>
<td>$r \sim_{\text{ACI}} r'$</td>
</tr>
<tr>
<td>6</td>
<td>$s \sim_{\text{ACI}} s'$</td>
</tr>
<tr>
<td>7</td>
<td>$\text{Conc } r s \sim_{\text{ACI}} \text{Conc } r' s'$</td>
</tr>
<tr>
<td>8</td>
<td>$r \sim_{\text{ACI}} s$</td>
</tr>
<tr>
<td>9</td>
<td>$s \sim_{\text{ACI}} r$</td>
</tr>
<tr>
<td>10</td>
<td>$\text{Star } r \sim_{\text{ACI}} \text{Star } r'$</td>
</tr>
<tr>
<td>11</td>
<td>$r \sim_{\text{ACI}} s$</td>
</tr>
<tr>
<td>12</td>
<td>$s \sim_{\text{ACI}} t$</td>
</tr>
<tr>
<td>13</td>
<td>$r \sim_{\text{ACI}} t$</td>
</tr>
</tbody>
</table>

**quotient_type** $\text{ACI } a \text{ re } = a \text{ re } / \sim_{\text{ACI}}$

**datatype** $\text{ldl } = \text{Prop } \text{string } \mid \text{And } \text{ldl } \text{ldl } \mid \text{Neg } \text{ldl } \mid \text{Match } (\text{ldl } \text{re}_{\text{ACI}})$

Unsupported recursive occurrence of type $\text{ldl}$ via type constructor $\text{re}_{\text{ACI}}$ in type expression $\text{ldl } \text{re}_{\text{ACI}}$.

Use the **bnf** command to register $\text{re}_{\text{ACI}}$ as a bounded natural functor to allow nested (co)recursion through it.
Interlude: Contribution

Identified sufficient conditions on when quotients of BNFs are BNFs

Relevant for (co)datatypes, relational parametricity, refinement
Interlude: Contribution

Identified sufficient conditions on when quotients of BNFs are BNFs

Relevant for (co)datatypes, relational parametricity, refinement

Automated BNF preservation proofs via lift_bnf command in
Datatype recursion worries

Unsupported recursive occurrence of type $ldl$ via type constructor $re_{ACI}$ in type expression $ldl \ re_{ACI}$.

Use the $\texttt{bnf}$ command to register $re_{ACI}$ as a bounded natural functor to allow nested (co)recursion through it.
Datatype recursion worries

Unsupported recursive occurrence of type \texttt{ldl} via type constructor\texttt{re}$_{\text{ACI}}$ in type expression \texttt{ldl re}$_{\text{ACI}}$.

Use the \texttt{bnf} command to register \texttt{re}$_{\text{ACI}}$ as a bounded natural functor to allow nested (co)recursion through it.

\begin{verbatim}
  **datatype** bad = C (bad set) | ... \\
  C :: bad set \Rightarrow bad  injective
\end{verbatim}
Unsupported recursive occurrence of type \( ldl \) via type constructor \( re_{ACI} \) in type expression \( ldl \, re_{ACI} \).

Use the \texttt{bnf} command to register \( re_{ACI} \) as a bounded natural functor to allow nested (co)recursion through it.

\textbf{Datatypes may recurse only through BNFs}

**Datatype recursion worries**

\textbf{Why We Can’t have SML Style datatype Declarations in HOL}

\begin{quote}
Elsa L. Gunter

AT&T Bell Laboratories, Rm. 2A-432, Murray Hill, NJ, 07974-0636, USA
\end{quote}

\begin{verbatim}
datatype bad = C (bad set) | ... C :: bad set ⇒ bad  injective
\end{verbatim}
Bounded Natural Functors (BNF)

```
F(A) = \{ [1], [], [2,3], [3,3], [1,2,3] \}
```

```
F:B → F(A)
```

```
F(1) = [1]
F(2) = []
F(3) = [2,3]
```
Bounded Natural Functors (BNF)

A

1 3
2

F

[1] []
[2,3] [3,3]
[1,2,3]

F(A)

B

a
b

F

[ ] [a]
[b,b] [a,b]
[a,b,b]

F(B)
Bounded Natural Functors (BNF)

Functor
\( \text{map}_F \text{id} = \text{id} \)
\( \text{map}_F \ g \circ \text{map}_F \ f = \text{map}_F \ (g \circ f) \)

\( F(A) \)
\( F(B) \)
Bounded Natural Functors (BNF)

Functor
\[ \text{map}_F \text{id} = \text{id} \]
\[ \text{map}_F g \circ \text{map}_F f = \text{map}_F (g \circ f) \]

Bound
\[ |\text{set}_F x| < \aleph \]

\[ F(A) \]
\[ \{1\}, \{2,3\}, \{1,2,3\} \]
\[ \text{map}_F f \]

\[ F(B) \]
\[ \{\}, \{3,3\}, \{a\} \]
\[ \{\}, \{b,b\}, \{a,b\} \]
\[ \{\}, \{a,b,b\} \]

A
\[ 1 \rightarrow 3 \]
\[ 2 \rightarrow b \]

B
\[ a \]

\[ F_{\text{set}_F} \]
Bounded Natural Functors (BNF)

**Functor**
- $\text{map}_F \text{id} = \text{id}$
- $\text{map}_F (g \circ \text{map}_F f) = \text{map}_F (g \circ f)$

**Bound**
- $|\text{set}_F x| < \aleph$

**Natural**
- $\text{set}_F (\text{map}_F f x) = f(\text{set}_F x)$
- $\forall x \in \text{set}_F x. f x = g x$
- $\text{map}_F f x = \text{map}_F g x$
Bounded Natural Functors (BNF)

Functor
\[ \text{map}_F \text{id} = \text{id} \]
\[ \text{map}_F g \circ \text{map}_F f = \text{map}_F (g \circ f) \]

Bound
\[ |\text{set}_F x| < \aleph \]

Natural
\[ \text{set}_F (\text{map}_F f x) = f(\text{set}_F x) \]
\[ \forall x \in \text{set}_F x. \text{f} x = \text{g} x \]
\[ \text{map}_F f x = \text{map}_F g x \]
Bounded Natural Functors (BNF)

**Functor**
\[
\text{map}_F \text{id} = \text{id} \\
\text{map}_F g \circ \text{map}_F f = \text{map}_F (g \circ f)
\]

**Bound**
\[
|\text{set}_F x| < \aleph
\]

**Natural**
\[
\text{set}_F (\text{map}_F f x) = f(\text{set}_F x) \\
\forall x \in \text{set}_F x. f x = g x \\
\text{map}_F f x = \text{map}_F g x
\]

**Relator**
\[
(x, y) \in \text{rel}_F R = \exists z \in F(R). \text{map}_F \pi_1 z = x \land \text{map}_F \pi_2 z = y \\
\text{rel}_F R \circ \text{rel}_F S = \text{rel}_F (R \circ S)
\]
Closure properties of BNF

Basic BNFs

_ + _  _ × _

τ ⇒ _

Non-BNFs

_ set _ ⇒ τ
Closure properties of BNF

Basic BNFs

- + _ _ × _

τ ⇒ _

Non-BNFs

_set _ _ ⇒ τ

Derived BNFs

composition

unit + _ _ × _

codatatypes

_ stream

datatypes

_ list

subtypes*

_ balanced-tree

* Conditions apply.
Viewing $re_{ACI}$ as a subtype

fun $nf_{ACI} :: a \Rightarrow a$ where ...

lemma $r \sim_{ACI} s \iff nf_{ACI} r = nf_{ACI} s$  \hspace{1cm} \langle proof \rangle

typedef $a \ re_{ACI} = \{ nf_{ACI} r | r :: a \ re \} \hspace{1cm} \text{by auto}$
Viewing $re_{\text{ACI}}$ as a subtype

fun $nf_{\text{ACI}} :: a \Rightarrow a$ where ...

lemma $r \sim_{\text{ACI}} s \iff nf_{\text{ACI}} r = nf_{\text{ACI}} s$  \hspace{1cm} \langle\text{proof}\rangle

typedef $a \, re_{\text{ACI}} = \{nf_{\text{ACI}} r \mid r :: a\}$ by auto

lift_bnf $a \, re_{\text{ACI}}$

1. $s \in NF \Rightarrow \text{map}_{\, re\,} f \, s \in NF$
2. ...
Viewing $re_{ACI}$ as a subtype

fun $nf_{ACI} :: a re \Rightarrow a re$ where ...

lemma $r \sim_{ACI} s \iff nf_{ACI} r = nf_{ACI} s$ \hspace{1cm} (proof)

typedef $a re_{ACI} = \{nf_{ACI} \mid r :: a re\} \; \text{by auto}$

lift_bnf $a re_{ACI}$

1. $s \in NF \longrightarrow \text{map}_{re} f \; s \in NF$
2. ...

unlikely for non-injective $f$
Viewing $re_{ACI}$ as a subtype

fun $nf_{ACI} :: a \, re \Rightarrow a \, re$ where ...

lemma $r \sim_{ACI} s \iff nf_{ACI} \, r = nf_{ACI} \, s \quad \langle proof \rangle$

typedef $a \, re_{ACI} = \left\{ nf_{ACI} \, r \mid r :: a \, re \right\}$ by auto

lift_bnf $a \, re_{ACI}$ unlikely for non-injective $f$

1. $s \in NF \implies \text{map}_{re} \, f \, s \in NF$
2. …

Quotients can be viewed as subtypes via representatives but we cannot lift the BNF structure along this view.
Data Types as Quotients of Polynomial Functors

Jeremy Avigad
Department of Philosophy, Carnegie Mellon University, Pittsburgh, PA, USA
http://www.andrew.cmu.edu/user/avigad/
avigad@cmu.edu

Mario Carneiro
Department of Philosophy, Carnegie Mellon University, Pittsburgh, PA, USA
di.gama@gmail.com

Simon Hudon
Department of Philosophy, Carnegie Mellon University, Pittsburgh, PA, USA
https://www.cmu.edu/dietrich/philosophy/people/postdoc-fellows/simon-hudon30.html
simon.hudon@gmail.com

Abstract

A broad class of data types, including arbitrary nestings of inductive types, coinductive types, and quotients, can be represented as quotients of polynomial functors. This provides perspicuous ways of constructing them and reasoning about them in an interactive theorem prover.
Data Types as Quotients of Polynomial Functors

Jeremy Avigad
Department of Philosophy, Carnegie Mellon University, Pittsburgh, PA, USA
http://www.andrew.cmu.edu/user/avigad/
avigad@cmu.edu

Mario Carneiro
Department of Philosophy, Carnegie Mellon University, Pittsburgh, PA, USA
di.gama@gmail.com

Simon Hudon
Department of Philosophy, Carnegie Mellon University, Pittsburgh, PA, USA
https://www.cmu.edu/dietrich/philosophy/people/postdoc-fellows/simon-hudon%20.html
simon.hudon@gmail.com

Abstract
A broad class of data types, including arbitrary nestings of inductive types, coinductive types, and quotients, can be represented as quotients of polynomial functors. This provides perspicuous ways of constructing them and reasoning about them in an interactive theorem prover.

1 Introduction
Data types are fundamental to programming, and theoretical computer science provides abstract characterizations of such data types and principles for reasoning about them. For example, an inductive type, such as the type of lists of elements of type \texttt{–}, is freely generated by its constructors:

\begin{verbatim}
inductive list (– : Type) | nil : list | cons : – ÷ list ÷ list 
\end{verbatim}

Such a declaration gives rise to a type constructor, \texttt{list}, constructors \texttt{nil} and \texttt{cons}, and a recursor:

\begin{verbatim}
list. rec {–—} : — ÷ (– ÷ list ÷ — ÷ —) ÷ list ÷ — ÷ —
\end{verbatim}

The recursor satisfies the following equations:

\begin{verbatim}
list. rec b f nil = b
list. rec b f (cons a l) = f a l (list. rec b f l)
\end{verbatim}

We also have an induction principle:

\begin{verbatim}
' {–} (P : list ÷ Prop), P nil ÷ (a l, P l ÷ P (cons a l)) ÷ ' a l, P l
\end{verbatim}

Supplement Material
Lean formalizations are online at https://github.com/avigad/qpf.

Funding
Work partially supported by AFOSR grant FA9550-18-1-0120 and the Sloan Foundation.

Acknowledgements
We are grateful to Andrei Popescu, Dmitriy Traytel, and Jasmin Blanchette for extensive discussions and very helpful advice.

© Jeremy Avigad, Mario Carneiro, and Simon Hudon; licensed under Creative Commons License CC-BY
Data Types as Quotients of Polynomial Functors

Jeremy Avigad
Department of Philosophy, Carnegie Mellon University, Pittsburgh, PA, USA
http://www.andrew.cmu.edu/user/avigad/
avigad@andrew.cmu.edu

Mario Carneiro
Department of Philosophy, Carnegie Mellon University, Pittsburgh, PA, USA
di.gama@gmail.com

Simon Hudon
Department of Philosophy, Carnegie Mellon University, Pittsburgh, PA, USA
https://www.cmu.edu/dietrich/philosophy/people/postdoc-fellows/simon-hudon%20.html
simon.hudon@gmail.com

Abstract

A broad class of data types, including arbitrary nestings of inductive types, coinductive types, and
quotients, can be represented as quotients of polynomial functors. This provides perspicuous ways
of constructing them and reasoning about them in an interactive theorem prover.

2012 ACM Subject Classification

Theory of computation

æ

Logic and verification; Theory of
computation

æ

Type theory; Theory of computation

æ

Data structures design and analysis

Keywords and phrases

data types, polynomial functors, inductive types, coinductive types

Digital Object Identifier

10.4230/LIPIcs.ITP.2019.6

Supplement Material

Lean formalizations are online at
https://github.com/avigad/qpf

Funding

Work partially supported by AFOSR grant FA9550-18-1-0120 and the Sloan Foundation.

Acknowledgements

We are grateful to Andrei Popescu, Dmitriy Traytel, and Jasmin Blanchette for
extensive discussions and very helpful advice.

1

Introduction

Data types are fundamental to programming, and theoretical computer science provides
abstract characterizations of such data types and principles for reasoning about them. For
example, an inductive type, such as the type of lists of elements of type –, is freely generated
by its constructors:

\[
\text{inductive \ list (–)} :: \text{Type} | \text{nil} : \text{list} | \text{cons} : \text{–} \rightarrow \text{list} \rightarrow \text{list}
\]

Such a declaration gives rise to a type constructor, \(\text{list}\), constructors \(\text{nil}\) and \(\text{cons}\), and a
recursor:

\[
\text{list}.\text{rec} \{ \text{–} \} : \text{–} \rightarrow (\text{–} \rightarrow \text{list} \rightarrow \text{list}) \rightarrow \text{list} \rightarrow \text{–}
\]

The recursor satisfies the following equations:

\[
\text{list}.\text{rec} b f \text{nil} = b
\]

\[
\text{list}.\text{rec} b f (\text{cons} a l) = f a l (\text{list}.\text{rec} b f l)
\]

We also have an induction principle:

\[
\{ \text{–} \} (P : \text{list} \rightarrow \text{Prop}), P \text{nil} \leftrightarrow (a l, P l \mapsto P (\text{cons} a l)) \leftrightarrow l, P l
\]

The diagram illustrates the mapping of \(\sim\) and the preservation of wide intersections and weak pullbacks.
A broad class of data types, including arbitrary nestings of inductive types, coinductive types, and quotients, can be represented as quotients of polynomial functors. This provides perspicuous ways of constructing them and reasoning about them in an interactive theorem prover.

**Abstract**
A broad class of data types, including arbitrary nestings of inductive types, coinductive types, and quotients, can be represented as quotients of polynomial functors. This provides perspicuous ways of constructing them and reasoning about them in an interactive theorem prover.

\[
x \sim y \longrightarrow \text{map}_F f \ x \sim \text{map}_F f \ y
\]

\[
x \sim y \longrightarrow \text{set}_F x = \text{set}_F y
\]
Quotients of Polynomial Functors

Data Types as Quotients of Polynomial Functors
Jeremy Avigad
Department of Philosophy, Carnegie Mellon University, Pittsburgh, PA, USA
http://www.andrew.cmu.edu/user/avigad/
avigad@cmu.edu
Mario Carneiro
Department of Philosophy, Carnegie Mellon University, Pittsburgh, PA, USA
di.gama@gmail.com
Simon Hudon
Department of Philosophy, Carnegie Mellon University, Pittsburgh, PA, USA
https://www.cmu.edu/dietrich/philosophy/people/postdoc-fellows/simon-hudon%20.html
simon.hudon@gmail.com

Abstract
A broad class of data types, including arbitrary nestings of inductive types, coinductive types, and
quotients, can be represented as quotients of polynomial functors. This provides perspicuous ways
of constructing them and reasoning about them in an interactive theorem prover.

2012 ACM Subject Classification
Theory of computation
æ Logic and verification; Theory of computation
æ Type theory; Theory of computation
æ Data structures design and analysis

Keywords and phrases
data types, polynomial functors, inductive types, coinductive types

Digital Object Identifier
10.4230/LIPIcs.ITP.2019.6

Supplement Material
Lean formalizations are online at https://github.com/avigad/qpf.

Funding
Work partially supported by AFOSR grant FA9550-18-1-0120 and the Sloan Foundation.

Acknowledgements
We are grateful to Andrei Popescu, Dmitriy Traytel, and Jasmin Blanchette for
extensive discussions and very helpful advice.

1 Introduction
Data types are fundamental to programming, and theoretical computer science provides
abstract characterizations of such data types and principles for reasoning about them. For
example, an inductive type, such as the type of lists of elements of type –, is freely generated
by its constructors:

\[
\text{inductive list (}–: \text{Type)} |
\text{nil: list} |
\text{cons:} – \rightarrow \text{list} \rightarrow \text{list}
\]

Such a declaration gives rise to a type constructor, list, constructors nil and cons, and a
recursor:

\[
\text{list. rec} \{–--\} : \text{–} \rightarrow (\text{–} \rightarrow \text{list} \rightarrow \text{list} \rightarrow \text{list}) \rightarrow \text{list}
\]

The recursor satisfies the following equations:

\[
\text{list. rec} b f \text{nil} = b \text{list. rec} b f (\text{cons} a l) = f a l (\text{list. rec} b f l)
\]

We also have an induction principle:

\[
'\{–\} (P : \text{list} \rightarrow \text{Prop}), P \text{nil} \rightarrow (a l, P l \rightarrow P (\text{cons} a l)) \rightarrow 'a l, P l
\]

© Jeremy Avigad, Mario Carneiro, and Simon Hudon; licensed under Creative Commons License CC-BY
10th International Conference on Interactive Theorem Proving (ITP 2019).
Editors: John Harrison, John O’Leary, and Andrew Tolmach; Article No. 6; pp. 6:1–6:19
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

\[
\sim \rightarrow \sim \map_F f x \sim \map_F f y
\]

\[
x \sim y \rightarrow \text{set}_F x = \text{set}_F y
\]

~ preserves wide intersections
~ preserves weak pullbacks

\[
\sim \rightarrow \map_F f \xymatrix{\text{set}_F x \sim \map F f y} \rightarrow \sim
\]

\[
\sim \rightarrow \map F f \xymatrix{\text{set}_F x \sim \map F f y} \rightarrow \sim
\]

\[
\sim \rightarrow \map F f \xymatrix{\text{set}_F x \sim \map F f y} \rightarrow \sim
\]

\[
\sim \rightarrow \map F f \xymatrix{\text{set}_F x \sim \map F f y} \rightarrow \sim
\]
Distinct Lists

\[
\begin{array}{cc}
[&] & [a] \\
[b,b] & [a,b] \\
[a,b,b] & \\
\end{array}
\]

typedef \text{a dlist} = \{ xs :: a list | \text{distinct} \, xs \} 

map (\lambda \_ \cdot a) 

\begin{array}{c}
\text{quotient type} \\
\text{a dlist} = \\
\text{a list} / \quad \lambda \, xs \, ys. \ \text{remdups} \, xs = \text{remdups} \, ys \\
\end{array}

\sim \quad \text{dlist} \\
\sim \quad \text{dlist} \quad \text{preserves wide intersections} \\
\begin{array}{c}
xs \sim \text{dlist} \, ys \\
\rightarrow \\
\text{set} \, xs = \text{set} \, ys \\
\end{array}

\sim \quad \text{dlist} \quad \text{preserves weak pullbacks}
Distinct Lists

typedef a dlist =
{xs :: a list | distinct xs}
Distinct Lists

<typename> a dlist =
{xs :: a list | distinct xs}
**Distinct Lists**

**typedef** a dlist = 
\{xs :: a list | distinct xs\}

**quotient_type** a dlist = 
a list / (\(\lambda xs\ ys. \text{remdups}\) xs = \(\text{remdups}\) ys)

\(\sim_d\)
Distinct Lists

**typedef** a dlist = 
{xs :: a list | distinct xs}

**quotient_type** a dlist =

a list / \( \lambda xs \, ys. \) remdups \( xs = \) remdups \( ys \)

\( \sim_{\text{dlist}} \)

- \( xs \sim_{\text{dlist}} ys \longrightarrow \) set \( xs = \) set \( ys \)
- \( \sim_{\text{dlist}} \) preserves wide intersections
- \( \sim_{\text{dlist}} \) preserves weak pullbacks
Terminated Lazy Lists

codatatype \( \text{a llist} = \text{LNil} \mid \text{LCons a (a llist)} \)
Terminated Lazy Lists

codatatype \( a \ llist = \text{LNil} \mid \text{LCons} \ a \ (a \ llist) \)

codatatype \( (a, b) \ tllist = \text{TLNil} \ b \mid \text{TLCons} \ a \ ((a, b) \ tllist) \)
Terminated Lazy Lists

**codatatype** \(a \mathit{llist} = \mathsf{LNil} \mid \mathsf{LCons} \ a \ (a \ \mathit{llist})\)

**codatatype** \((a, b) \ \mathit{tllist} = \mathsf{TLNil} \ b \mid \mathsf{TLCons} \ a \ ((a, b) \ \mathit{tllist})\)
Terminated Lazy Lists

codatatype $a \text{ llist} = \text{LNil} \mid \text{LCons } a \ (a \text{ llist})$

codatatype $(a, b) \text{ tllist} = \text{TLNil } b \mid \text{TLCons } a \ ((a, b) \text{ tllist})$
Terminated Lazy Lists

**codatatype** \( a \ llist = \text{LNil} \mid \text{LCons} \ a \ (a \ llist) \)

**codatatype** \((a, b) \ tllist = \text{TLNil} \ b \mid \text{TLCons} \ a \ ((a, b) \ tllist)\)
Terminated Lazy Lists

codatatype \( a \) llist = LNil \( \mid \) LCons \( a \) (\( a \) llist)

codatatype (\( a, b \)) tllist = TLNil \( b \) \( \mid \) TLCons \( a \) ((\( a, b \)) tllist)
Terminated Lazy Lists

codatatype \( a \text{ llist} = \text{LNil} \mid \text{LCons} \ a \ (a \text{ llist}) \)

quotient_type \( (a, b) \ tllist = \)
\[
\text{a llist} \times \text{b} \ /
(\lambda (xs,\alpha) (ys,\beta). \ xs = \ ys \land (|xs| < \infty \rightarrow \alpha = \beta)) \)

\( \sim_{tllist} \)
Terminated Lazy Lists

codatatype \( a \, llist = \) LNil | LCons \( a \, (a \, llist) \)

quotient_type \( (a, b) \, tllist = \)
\( a \, llist \times b / (\lambda (xs, \alpha) (ys, \beta). \, xs = ys \land (|xs| < \infty \rightarrow \alpha = \beta)) \)

\((xs, \alpha) \sim_{tllist} (ys, \beta) \rightarrow set_{llist} \, xs = set_{llist} \, ys\)

\((xs, \alpha) \sim_{tllist} (ys, \beta) \rightarrow \{\alpha\} = \{\beta\}\)

~\(_{tllist}\) preserves wide intersections

~\(_{tllist}\) preserves weak pullbacks
Terminated Lazy Lists

codatatype \( a \) llist = LNil | LCons \( a \) (a llist)

quotient\_type \((a, b)\) tllist =
\( a \) llist \( \times \) b / \((\lambda (xs, \alpha)(ys, \beta). xs = ys \land (|xs| < \infty \rightarrow \alpha = \beta))\)

- \( (xs, \alpha) \sim tllist (ys, \beta) \rightarrow \text{set}_{llist} xs = \text{set}_{llist} ys \)
- \( (xs, \alpha) \sim tllist (ys, \beta) \rightarrow \{\alpha\} = \{\beta\} \)
- \( \sim_{tllist} \) preserves wide intersections
- \( \sim_{tllist} \) preserves weak pullbacks
How to correct?
How to correct?
How to correct?

```plaintext
datatype a option = None | Some a
```

Diagram showing the mapping of `F(A)` to `F(Some[A])` and the operation `map_F Some`.
How to correct?

datatype \( a \) option = None | Some \( a \)
How to correct?

datatype \( a \) option = None | Some \( a \)

\[
\text{set}_F/\sim [\{x\}] = \bigcap_{y \in [\text{map}_F \text{Some } x]} \{a. \text{Some } a \in \text{set}_F y\}
\]

\( F(\{\text{None}\} \cup \text{Some}[A])/\sim \)

\( \sim \)
Preservation theorem

- BNF $F$ with equivalence relation $\sim$
- $\sim$ preserves wide intersections
  \[ A \neq \{\} \land \bigcap A \neq \{\} \longrightarrow \bigcap \{[A]_\sim | A \in F(A)\} \subseteq \bigcap F(A)_\sim \]
- $\sim$ weakly preserve pullbacks
  \[ R \cdot S \neq \{\} \longrightarrow \text{rel}_F R \cdot \sim \cdot \text{rel}_F S \subseteq \sim \cdot \text{rel}_F (R \cdot S) \cdot \sim \]
Preservation theorem

- BNF F with equivalence relation ∼
- ∼ preserves wide intersections
  \[ \mathcal{A} \neq \emptyset \land \bigcap \mathcal{A} \neq \emptyset \longrightarrow \bigcap \{[A]_\sim | A \in F(\mathcal{A})\} \subseteq [\bigcap F(\mathcal{A})]_\sim \]
- ∼ weakly preserve pullbacks
  \[ R \cdot S \neq \emptyset \longrightarrow \text{rel}_F R \cdot \sim \cdot \text{rel}_F S \subseteq \sim \cdot \text{rel}_F (R \cdot S) \cdot \sim \]

yields BNF for F/∼

- \[ \text{map}_{F/\sim} f [x]_\sim = [\text{map}_F f \ x]_\sim \]
- \[ \text{set}_{F/\sim} [x]_\sim = \bigcap_{y \in [\text{map}_F \ Some \ x]_\sim} \{ a. \ Some \ a \in \text{set}_F y \} \]
- \([x]_\sim [y]_\sim \) ∈ rel\(_{F/\sim}\) R ⇔ \((\text{map}_F \ Some \ x, \text{map}_F \ Some \ y) \) ∈ (\sim \cdot \text{rel}_F (\text{rel}_{option} R) \cdot \sim)\]
**lift_bnf** in action

codatatype $a \text{l}l\text{i}s\text{t} = \text{LNil} \mid \text{LCons} \ a \ (a \ l\text{i}s\text{t})$

definition $\sim_{\text{tllist}} :: a \text{l}l\text{i}s\text{t} \times b \Rightarrow a \text{l}l\text{i}s\text{t} \times b \Rightarrow \text{bool}$ where

$(xs, \alpha) \sim_{\text{tllist}} (ys, \beta) \iff xs = ys \land (|xs| < \infty \implies \alpha = \beta)$

quotient_type $(a, b) \ tllist = a \text{l}l\text{i}s\text{t} \times b / \sim_{\text{tllist}}$

lift_bnf $(a, b) \ tllist$
**lift_bnf in action**

**codatatype**  
\[ a llist = \text{LNil} \mid \text{LCons} a (a llist) \]

**definition**  
\[ \sim_{\text{tllist}} :: a llist \times b \Rightarrow a llist \times b \Rightarrow \text{bool} \quad \text{where} \]
\[ (xs, \alpha) \sim_{\text{tllist}} (ys, \beta) \iff xs = ys \land (|xs| < \infty \implies \alpha = \beta) \]

**quotient_type**  
\[ (a, b) \, \text{tllist} = a llist \times b / \sim_{\text{tllist}} \]

**lift_bnf**  
\[ (a, b) \, \text{tllist} \]

1. \[ A \cdot A' \neq \perp \quad \implies \quad B \cdot B' \neq \perp \quad \implies \quad \text{rel}_x (\text{rel}_{\text{tllist}} A) B \cdot \sim_{\text{tllist}} \cdot \text{rel}_x (\text{rel}_{\text{tllist}} A') B' \leq \sim_{\text{tllist}} \cdot \text{rel}_x (\text{rel}_{\text{tllist}} (A \cdot A')) (B \cdot B') \cdot \sim_{\text{tllist}} \]

2. \[ S \neq \{\} \quad \implies \quad \bigcap S \neq \{\} \quad \implies \quad \bigcap \{x. \exists y. y \sim_{\text{tllist}} x \land \text{set}_{\text{llist}} (\pi_1 y) \subseteq A\} \subseteq \{x. \exists y. y \sim_{\text{tllist}} x \land \text{set}_{\text{llist}} (\pi_1 y) \subseteq \bigcap S\} \quad A \in S \]

3. \[ S \neq \{\} \quad \implies \quad \bigcap S \neq \{\} \quad \implies \quad \bigcap \{x. \exists y. y \sim_{\text{tllist}} x \land \pi_2 y \in A\} \subseteq \{x. \exists y. y \sim_{\text{tllist}} x \land \pi_2 y \in \bigcap S\} \quad A \in S \]
lift_bnf in action

codatatype \( a \) llist = LNil | LCons \( a \) (a llist)

definition \( \sim _{tllist} : a \) llist \( \times b \Rightarrow a \) llist \( \times b \Rightarrow \) bool where
\( (xs, \alpha) \sim_{tllist} (ys, \beta) \leftrightarrow xs = ys \land (|xs| < \infty \rightarrow \alpha = \beta) \)

quotient_type \((a, b)\) tllist = \( a \) llist \( \times b \) / \( \sim _{tllist} \)

lift_bnf \((a, b)\) tllist

subgoal by (auto 0 4 simp: \( \sim _{tllist} \) _def …)
subgoal by (auto simp: \( \sim _{tllist} \) _def)
subgoal by (auto 6 0 simp: \( \sim _{tllist} \) _def)
done
lift_bnf in action

codatatype \( a \) \textit{llist} = \textit{LNil} | \textit{LCons} \( a \) \( a \) \textit{llist}

definition \( \sim \text{tl}l\text{ist} :: \textit{a llist} \times b \Rightarrow \textit{a llist} \times b \Rightarrow \text{bool} \) where
\[
(x_s, \alpha) \sim_{\text{tl}l\text{ist}} (y_s, \beta) \iff x_s = y_s \land (|x_s| < \infty \implies \alpha = \beta)
\]

quotient_type \( (a, b) \text{tl}l\text{ist} = \textit{a llist} \times b \) / \( \sim_{\text{tl}l\text{ist}} \)

lift_bnf \( (a, b) \text{tl}l\text{ist} \)

subgoal by \( \text{(auto 0 4 simp: } \sim_{\text{tl}l\text{ist} \_def} \ldots) \)

subgoal by \( \text{(auto simp: } \sim_{\text{tl}l\text{ist} \_def}) \)

subgoal by \( \text{(auto 6 0 simp: } \sim_{\text{tl}l\text{ist} \_def}) \)

done

datatype \textit{foo} = \textit{E} | \textit{C} ((\textit{foo, foo}) \text{tl}l\text{ist})
x \rel_{F} R y ~ y' \rel_{F} S z
\( u \in F(R) \)
\[ x \xrightarrow{\text{rel}_F \ R} y \sim y' \xrightarrow{\text{rel}_F \ S} z \]

\[ u \in F(R) \]

\[ v \in F(S) \]
\[
s \xrightarrow{\text{map}_F \pi_1} y \xrightarrow{\sim} y' \xrightarrow{\text{map}_F \pi_2} z
\]

\[
x \xrightarrow{\text{rel}_F R} y \xrightarrow{\text{map}_F \pi_1} u \in F(R)
\]

\[
y \xrightarrow{\text{map}_F \pi_2} y' \xrightarrow{\text{rel}_F S} z \xrightarrow{\text{map}_F \pi_2} v \in F(S)
\]
\( u \in F(R) \)

\( v \in F(S) \)

\( x \xrightarrow{\text{rel}_F R} y \xrightarrow{\sim} y' \xrightarrow{\text{rel}_F S} z \)

\( u' \in F(R) \)

\( w \)

\( \text{map}_F \pi_1 \)

\( \text{map}_F \pi_2 \)

\( \sim \)

\( * \)
\[
\begin{array}{c}
x \xrightarrow{\text{rel}_F \mathcal{R}} y \\
\map_F \pi_1 \quad \map_F \pi_2
\end{array}
\quad ~ \quad
\begin{array}{c}
\sim \\
u \in \mathcal{F}(R)
\end{array}
\quad ~ \quad
\begin{array}{c}
y' \xrightarrow{\text{rel}_F \mathcal{S}} z \\
\map_F \pi_1 \quad \map_F \pi_2
\end{array}
\quad ~ \quad
\begin{array}{c}
v \in \mathcal{F}(S)
\end{array}
\]
\[ x \xrightarrow{\text{rel}_F R} y \xrightarrow{\sim} y' \xrightarrow{\text{rel}_F S} z \]

\[ u \in F(R), \quad v \in F(S) \]

\[ u' \in F(R), \quad v' \in F(S) \]

\[ x' \quad w \quad z' \]

\[ \text{map}_F \pi_1 \quad \text{map}_F \pi_2 \]

\[ \text{map}_F \pi_1 \quad \text{map}_F \pi_2 \]
\[
\begin{align*}
\text{rel}_F R & \quad \sim \quad \text{rel}_F S \\
x & \quad \map_{F \pi_1} \quad \map_{F \pi_2} \quad y \\
& \quad \map_{F \pi_1} \quad \map_{F \pi_2} \quad y' \\
& \quad \map_{F \pi_1} \quad \map_{F \pi_2} \quad z \\
x' & \quad \map_{F \pi_1} \quad \map_{F \pi_2} \quad w \\
& \quad \map_{F \pi_1} \quad \map_{F \pi_2} \quad v' \\
& \quad \map_{F \pi_1} \quad \map_{F \pi_2} \quad z' \\
\end{align*}
\]
\[ u \in F(R) \]

\[ v \in F(S) \]

\[ x \sim y \sim z \]

\[ u' \in F(R) \]

\[ v' \in F(S) \]

\[ x' \sim y' \sim z' \]

\[ \text{rel}_F (R \bullet S) \]
Subdistributivity via rewrite relation

**Sufficient conditions:**

- BNF $F$ with equivalence relation $\sim$
- $x \sim y \rightarrow \text{map}_F f \ x \sim \text{map}_F f \ y \land \text{set}_F \ x = \text{set}_F \ y$
Subdistributivity via rewrite relation

**Sufficient conditions:**

- BNF $F$ with equivalence relation $\sim$
- $x \sim y \implies \text{map}_F f x \sim \text{map}_F f y \land \text{set}_F x = \text{set}_F y$
- Rewrite relation $\leadsto$ over-approximates $\sim$

- $\leadsto$ confluent: $y \rightsquigarrow z$ and factors through projections:

$\text{map}_F \pi_i$

Proof effort: 50% shorter (58 instead of 126 lines)
Subdistributivity via rewrite relation

**Sufficient conditions:**

- BNF F with equivalence relation $\sim$
- $x \sim y \rightarrow \text{map}_F f \, x \sim \text{map}_F f \, y \land \text{set}_F \, x = \text{set}_F \, y$
- Rewrite relation $\rightsquigarrow$ over-approximates $\sim$
- $\rightsquigarrow$ confluent: $y \rightsquigarrow z$ and factors through projections:
  
  - Distinct lists: $\text{xs} \cdot \text{ys} \rightsquigarrow \text{xs} \cdot [x] \cdot \text{ys}$ if $x \in \text{ys}$
  - Proof effort: 50% shorter (58 instead of 126 lines)
**a re_{ACI} is a BNF**

**inductive \( \rightsquigarrow_{ACI} \) where**

\[
\begin{align*}
  r & \rightsquigarrow_{ACI} r' \quad s \rightsquigarrow_{ACI} s' \\
  \text{Alt } r \; s & \rightsquigarrow_{ACI} \text{Alt } r' \; s'
\end{align*}
\]

\[
\begin{align*}
  r & \rightsquigarrow_{ACI} r' \quad s \rightsquigarrow_{ACI} s' \\
  \text{Conc } r \; s & \rightsquigarrow_{ACI} \text{Conc } r' \; s'
\end{align*}
\]

\[
\begin{align*}
  r & \rightsquigarrow_{ACI} r' \\
  \text{Star } r & \rightsquigarrow_{ACI} \text{Star } r'
\end{align*}
\]

\[
\begin{align*}
  r & \rightsquigarrow_{ACI} r \\
  r & \rightsquigarrow_{ACI} \text{Alt } r \; r \\
  \text{Alt } r \; s & \rightsquigarrow_{ACI} \text{Alt } s \; r \\
  \text{Alt } (\text{Alt } r \; s) & \rightsquigarrow_{ACI} \text{Alt } r \; (\text{Alt } s \; t) \\
  \text{Alt } r \; (\text{Alt } s \; t) & \rightsquigarrow_{ACI} \text{Alt } (\text{Alt } r \; s) \; t
\end{align*}
\]
**Inductive** $\Rightarrow_{\text{ACI}}$ **where**

- $r \Rightarrow_{\text{ACI}} r' s \Rightarrow_{\text{ACI}} s'$
- $\text{Alt } r s \Rightarrow_{\text{ACI}} \text{Alt } r' s'$
- $\text{Conc } r s \Rightarrow_{\text{ACI}} \text{Conc } r' s'$
- $r \Rightarrow_{\text{ACI}} r'$
- $\text{Star } r \Rightarrow_{\text{ACI}} \text{Star } r'$
- $r \Rightarrow_{\text{ACI}} r$
- $r \Rightarrow_{\text{ACI}} \text{Alt } r r$
- $\text{Alt } r s \Rightarrow_{\text{ACI}} \text{Alt } s r$
- $\text{Alt } (\text{Alt } r s) t \Rightarrow_{\text{ACI}} \text{Alt } r (\text{Alt } s t)$
- $\text{Alt } r (\text{Alt } s t) \Rightarrow_{\text{ACI}} \text{Alt } (\text{Alt } r s) t$

\[
\text{a re}_{\text{ACI}} \text{ is a BNF}
\]

- $\Rightarrow_{\text{ACI}} \cup \Rightarrow_{\text{ACI}}^{-1} = (\sim_{\text{ACI}})$
- $\Rightarrow_{\text{ACI}}$ is confluent
- $\text{map}_{\text{re}} \pi_1 r \Rightarrow_{\text{ACI}} s \Longrightarrow \exists t. t \sim_{\text{ACI}} r \land s = \text{map}_{\text{re}} \pi_1 t$
- $\text{map}_{\text{re}} \pi_2 r \Rightarrow_{\text{ACI}} s \Longrightarrow \exists t. t \sim_{\text{ACI}} r \land s = \text{map}_{\text{re}} \pi_2 t$
**a re**_{\text{ACI}} \text{ is a BNF}

- $(\Rightarrow_{\text{ACI}} \cup \Rightarrow_{\text{ACI}})^* = (\sim_{\text{ACI}})$
- $\Rightarrow_{\text{ACI}}$ is confluent
- $\text{map}_{\text{re}} \pi_1 r \Rightarrow_{\text{ACI}} s \rightarrow \exists t. t \sim_{\text{ACI}} r \land s = \text{map}_{\text{re}} \pi_1 t$
- $\text{map}_{\text{re}} \pi_2 r \Rightarrow_{\text{ACI}} s \rightarrow \exists t. t \sim_{\text{ACI}} r \land s = \text{map}_{\text{re}} \pi_2 t$

**lift_bnf** a re_{\text{ACI}} \langle \text{proof} \rangle
a re\textsubscript{ACI} is a BNF

\[ (\rightsquigarrow_{\text{ACI}} \cup \rightsquigarrow_{\text{ACI}}^{-1})^* = (\sim_{\text{ACI}}) \]

- \(\rightsquigarrow_{\text{ACI}}\) is confluent

\[ \text{map}_{\text{re}} \pi_1 \ r \rightsquigarrow_{\text{ACI}} s \rightarrow \exists t. \ t \sim_{\text{ACI}} r \land s = \text{map}_{\text{re}} \pi_1 t \]

\[ \text{map}_{\text{re}} \pi_2 \ r \rightsquigarrow_{\text{ACI}} s \rightarrow \exists t. \ t \sim_{\text{ACI}} r \land s = \text{map}_{\text{re}} \pi_2 t \]

\textbf{lift\textunderscore bnf} \ a \textsubscript{reACI} \langle \text{proof} \rangle

\textbf{datatype} \ ldl = \ Prop \ string \mid \text{And} \ ldl \ ldl \mid \text{Neg} \ ldl \mid \text{Match} \ (ldl \ \text{re}_{\text{ACI}})
Epilogue

lift_bnf

- part of Isabelle2020
- 1600 lines of Isabelle/ML
- generation of transfer rules
Epilogue

**lift_bnf**
- part of Isabelle2020
- 1600 lines of Isabelle/ML
- generation of transfer rules

**Applications**
- (co)datatypes
- Lifting and Transfer
- QPF

Limitations
- terms modulo α-equivalence
  - [Blanchette, Gheri, Popescu, T., POPL'19]
- signed multisets
  - [Blanchette, Fleury, T., FSCD'16]

Future Work
- partial quotients
- generalizations of BNFs
  - [L., S., ITP'18]
  - [Blanchette, Gheri, Popescu, T., POPL'19]
Epilogue

**lift_bnf**
- part of Isabelle2020
- 1600 lines of Isabelle/ML
- generation of transfer rules

**Applications**
- (co)datatypes
- Lifting and Transfer
- QPF

**Limitations**
- terms modulo $\alpha$-equivalence
  [Blanchette, Gheri, Popescu, T., POPL'19]
- signed multisets
  [Blanchette, Fleury, T., FSCD'16]
Epilogue

lift_bnf

- part of Isabelle2020
- 1600 lines of Isabelle/ML
- generation of transfer rules

Applications

- (co)datatypes
- Lifting and Transfer
- QPF

Limitations

- terms modulo \(\alpha\)-equivalence
  [Blanchette, Gheri, Popescu, T., POPL'19]
- signed multisets
  [Blanchette, Fleury, T., FSCD'16]

Future Work

- partial quotients
- generalizations of BNFs
  [L., S., ITP'18]
  [Blanchette, Gheri, Popescu, T., POPL'19]
Quotients of Bounded Natural Functors

Basil Fürer
Andreas Lochbihler
Joshua Schneider
Dmitriy Traytel

merci!

questions?