Foundational Nonuniform (Co)datatypes for Higher-Order Logic

Jasmin Blanchette

VU VRIJE UNIVERSITEIT AMSTERDAM

Fabian Meier

ETH zürich

Andrei Popescu

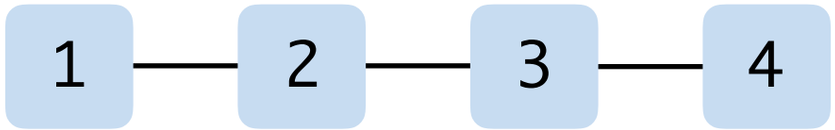
Middlesex University London

Dmitriy Traytel

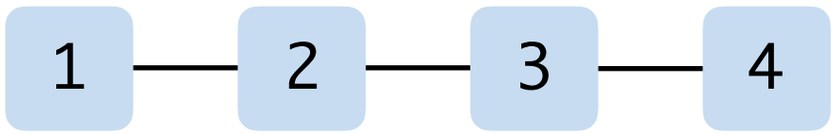
Isabelle HOL

ETH zürich

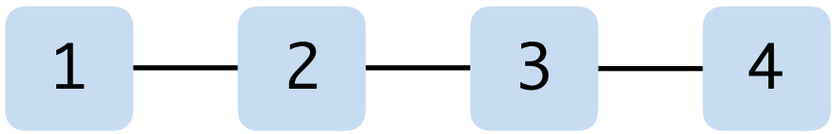
uniform datatype `'a list = Nil | Cons 'a ('a list)`



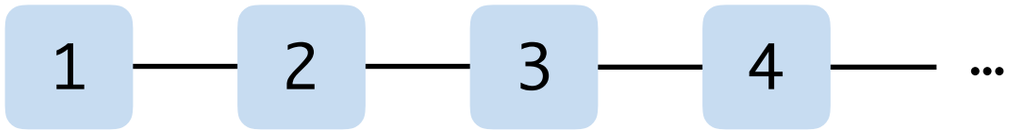
uniform datatype `'a list` = Nil | Cons 'a (`'a list`)



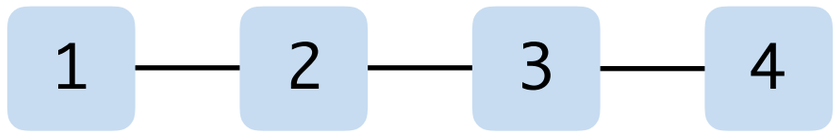
uniform datatype `'a list = Nil | Cons 'a ('a list)`



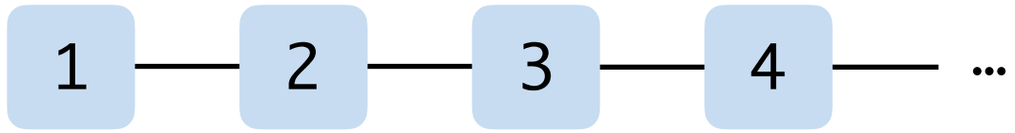
uniform codatatype `'a stream ≅ SCons 'a ('a stream)`



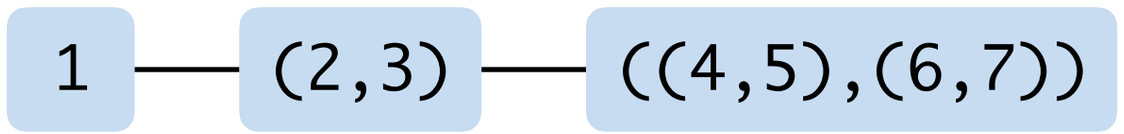
uniform datatype `'a list = Nil | Cons 'a ('a list)`



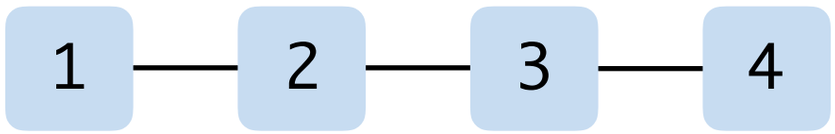
uniform codatatype `'a stream ≅ SCons 'a ('a stream)`



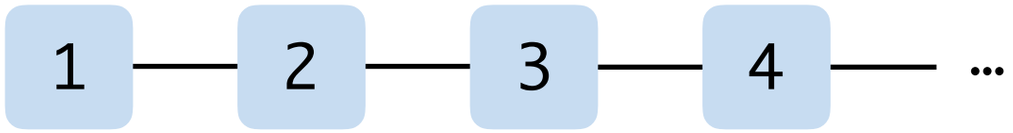
nonuniform datatype `'a plist = PNil | PCons 'a (('a × 'a) plist)`



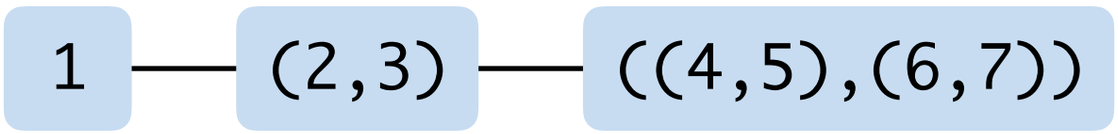
uniform datatype `'a list = Nil | Cons 'a ('a list)`



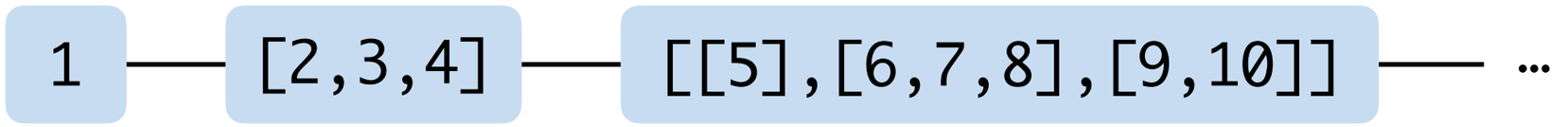
uniform codatatype `'a stream ≅ SCons 'a ('a stream)`



nonuniform datatype `'a plist = PNil | PCons 'a (('a × 'a) plist)`



nonuniform codatatype `'a pstream ≅ PSCons 'a (('a list) pstream)`



What are nonuniform types good for?

Mycroft
Okasaki

pioneering: optimization techniques

bootstrapping
implicit recursive slowdown

theory: data structures

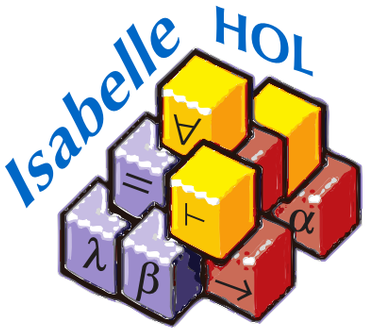
finger trees
generalized folds
advanced (co)iteration

Bird Paterson Hinze
Matthes Abel Uustalu
Abbott Altenkirch Ghani
…

practice: proof assistants

binders
balancing lists
finger trees
complexity

Benton Hur Kennedy McBride
Danielsson Hirschowitz Maggesi
Naves Spiwack Sozeau
…

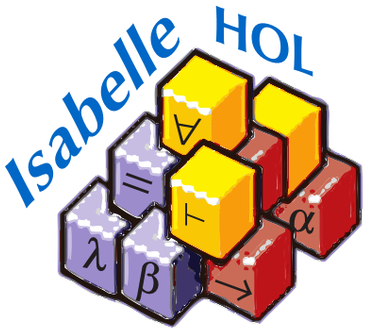
Contribution: enable users of 

to …

1. define nonuniform (co)datatypes

2. define primitively (co)recursive functions

3. prove theorems by nonuniform (co)induction

Contribution: enable users of 

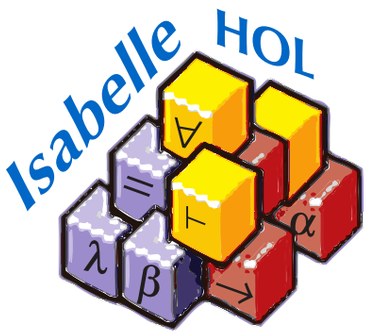
to …

1. define nonuniform (co)datatypes

```
'a tm = Var 'a | App ('a tm) ('a tm) | Lam (('a option) tm)
```

2. define primitively (co)recursive functions

3. prove theorems by nonuniform (co)induction

Contribution: enable users of 

to …

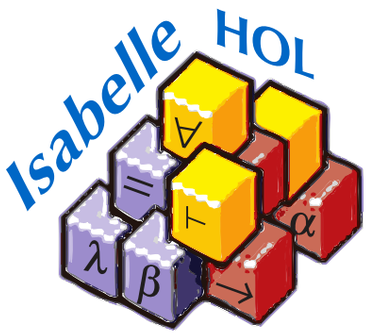
1. define nonuniform (co)datatypes

```
'a tm = Var 'a | App ('a tm) ('a tm) | Lam (('a option) tm)
```

2. define primitively (co)recursive functions

```
join :: 'a tm tm => 'a tm
join (Var t) = t
join (App t u) = App (join t) (join u)
join (Lam u) = Lam (join (map_tm
  (λx. case x of None => Var None | Some y => map_tm Some y) u))

subst σ = join ∘ map_tm σ
```

3. prove theorems by nonuniform (co)induction

Contribution: enable users of 

to …

1. define nonuniform (co)datatypes

```
'a tm = Var 'a | App ('a tm) ('a tm) | Lam (('a option) tm)
```

2. define primitively (co)recursive functions

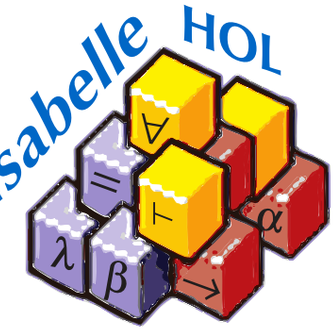
```
join :: 'a tm tm => 'a tm
join (Var t) = t
join (App t u) = App (join t) (join u)
join (Lam u) = Lam (join (mapₜₘ
  (λx. case x of None => Var None | Some y => mapₜₘ Some y) u))

subst σ = join ∘ mapₜₘ σ
```

3. prove theorems by nonuniform (co)induction

```
subst τ (subst σ s) = subst (subst τ ∘ σ) s
```

But Coq and Agda

have had this built into their logics for decades!



s of to …

tatypes

Lam (('a option) tm)

ecursive functions

join (Lam u) = Lam (join (map$_{tm}$
(λx. case x of None => Var None | Some y => map$_{tm}$ Some y) u))

subst σ = join ∘ map$_{tm}$ σ

3 prove theorems by nonuniform (co)induction

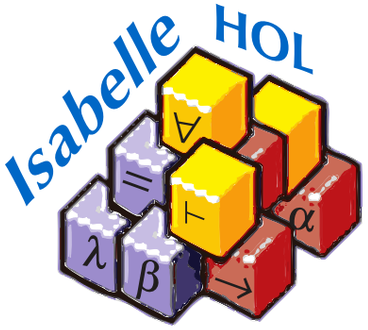
subst τ (subst σ s) = subst (subst τ ∘ σ) s

But Coq and Agda

have had this

into their lo

for decade

s of 

to …

```
join (Lam u) = Lam (join (
  (λx. case x of None => Va

subst σ = join ∘ map_tm σ
```

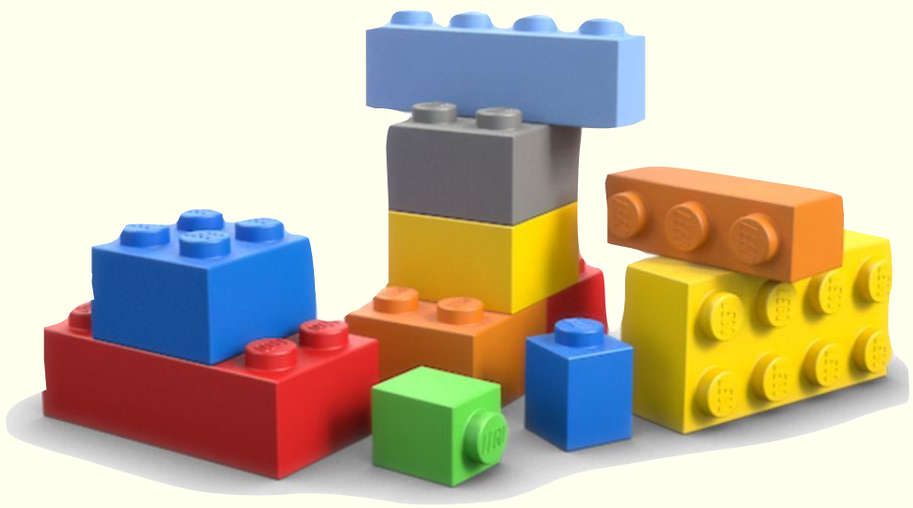
3 prove theorems by

```
subst τ (subst σ s) = subst
```

Our approach is foundational

new features are reduced to existing features

Foundations



Simple Theory of Types

Alonzo Church 1940

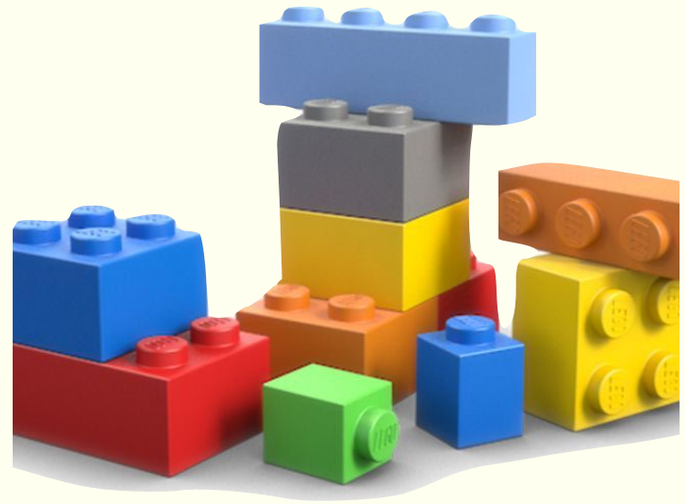
types: T = o | ι | T => T

terms: simply typed λ-calculus

+ few built-in constants

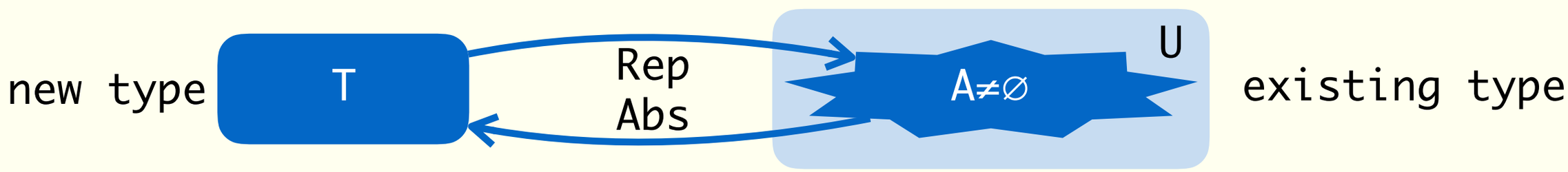
Higher-Order Logic

Mike Gordon 1988



types: T = o | ι | T => T | 'a | (T,…,T)κ

+ nonrecursive type definitions



terms: simply typed λ-calculus

+ few built-in constants

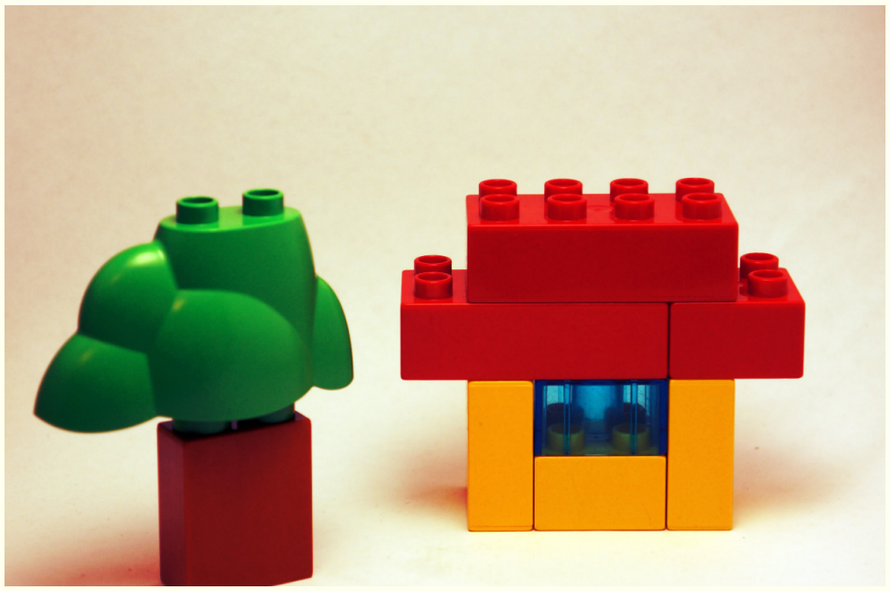
+ Hilbert Choice

+ nonrecursive constant definitions

Foundational Uniform (Co)datatypes for Higher-Order Logic

Jasmin Blanchette Andrei Popescu Dmitriy Traytel
et al.

LICS 2012 ITP 2014 ESOP 2015 ICFP 2015 ESOP 2017 FroCoS 2017



```
theory Examples
imports Library
begin

datatype_new 'a list = Nil | Cons 'a "'a list"
codatatype 'a llist = LNil | LCons 'a "'a llist"
```

```
theory Examples
imports Library
begin

datatype_new 'a list = Nil | Cons 'a "'a list"
codatatype 'a llist = LNil | LCons 'a "'a llist"

datatype_new 'a tree = Node 'a "'a tree list"
datatype_new 'a tree_ω = Node_ω 'a "'a tree_ω llist"
codatatype 'a ltree = LNode 'a "'a ltree list"
codatatype 'a ltree_ω = LNode_ω 'a "'a ltree_ω llist"
```

```
theory Examples
imports Library
begin

datatype_new 'a list = Nil | Cons 'a "'a list"
codatatype 'a llist = LNil | LCons 'a "'a llist"

datatype_new 'a tree = Node 'a "'a tree list"
datatype_new 'a tree_ω = Node_ω 'a "'a tree_ω llist"
codatatype 'a ltree = LNode 'a "'a ltree list"
codatatype 'a ltree_ω = LNode_ω 'a "'a ltree_ω llist"

datatype_new 'a tree_fset = Node_fset 'a "'a tree_fset fset"
codatatype 'a tree_cset = LNode_cset 'a "'a tree_cset cset"
```

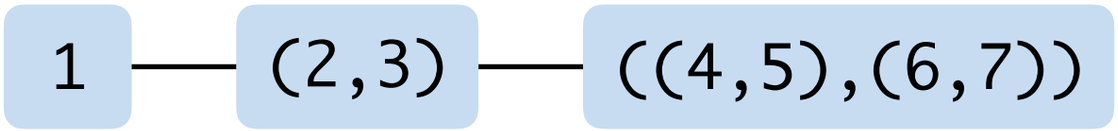
```
theory Examples
imports Library
begin

datatype_new 'a list = Nil | Cons 'a "'a list"
codatatype 'a llist = LNil | LCons 'a "'a llist"

datatype_new 'a tree = Node 'a "'a tree list"
datatype_new 'a tree_ω = Node_ω 'a "'a tree_ω llist"
codatatype 'a ltree = LNode 'a "'a ltree list"
codatatype 'a ltree_ω = LNode_ω 'a "'a ltree_ω llist"

datatype_new 'a tree_fset = Node_fset 'a "'a tree_fset fset"
codatatype 'a tree_cset = LNode_cset 'a "'a tree_cset cset"

datatype_new 'a tree_set = Node_set 'a "'a tree_set set"
```

```
Unsupported recursive occurrence of type
  "'a local.tree_set" via type constructor "Set.set" in
  type expression "'a local.tree_set set"
Use the "bnf" command to register "Set.set" as a bounded
  natural functor to allow nested (co)recursion through it
```

Foundational Nonuniform (Co)datatypes for Higher-Order Logic

LICS 2017

```
'a plist = PNil | PCons 'a (('a × 'a) plist)
```



1

2

3

4

```
'a plist = PNil | PCons 'a (('a × 'a) plist)
```

1 —— (2,3) —— ((4,5),(6,7))

1 overapproximate the elements of a powerlist

```
'a elem = Leaf 'a | Node ('a elem × 'a elem)
```

2

3

4

```
'a plist = PNil | PCons 'a (('a × 'a) plist)
```

1 —— (2,3) —— ((4,5),(6,7))

1 overapproximate the elements of a powerlist

```
'a elem = Leaf 'a | Node ('a elem × 'a elem)
```

2

3

4

```
'a plist = PNil | PCons 'a (('a × 'a) plist)
```

1 —— (2,3) —— ((4,5),(6,7))

1 overapproximate the elements of a powerlist

```
'a elem = Leaf 'a | Node ('a elem × 'a elem)
```

1 (2,3) ((4,5),(6,7))

2

3

4

```
'a plist = PNil | PCons 'a (('a × 'a) plist)
```

1 —— (2,3) —— ((4,5),(6,7))

1 overapproximate the elements of a powerlist

```
'a elem = Leaf 'a | Node ('a elem × 'a elem)
```

1 (2,3) ((4,5),(6,7)) ((4,5),6)

2

3

4

'a plist = PNil | PCons 'a (('a × 'a) plist)

1 — (2,3) — ((4,5),(6,7))

1 overapproximate the elements of a powerlist

'a elem = Leaf 'a | Node ('a elem × 'a elem)

1   (2,3)   ((4,5),(6,7))   ((4,5),6)

$$\frac{}{\texttt{full 0 (Leaf x)}} \qquad \frac{\texttt{full n l} \qquad \texttt{full n r}}{\texttt{full (n + 1) (Node (l, r))}}$$

2

3

4

```
'a plist = PNil | PCons 'a (('a × 'a) plist)
```

1 —— (2,3) —— ((4,5),(6,7))

1 overapproximate the elements of a powerlist

```
'a elem = Leaf 'a | Node ('a elem × 'a elem)
```

1 (2,3) ((4,5),(6,7)) ((4,5),6)

$$\frac{}{\texttt{full 0 (Leaf x)}} \qquad \frac{\texttt{full n l} \qquad \texttt{full n r}}{\texttt{full (n + 1) (Node (l, r))}}$$

2 overapproximate the set of all powerlists

```
'a plist₀ = PNil₀ | PCons₀ ('a elem) ('a plist₀)
```

3

4

```
'a plist = PNil | PCons 'a (('a × 'a) plist)
```

1 — (2,3) — ((4,5),(6,7))

1 overapproximate the elements of a powerlist

```
'a elem = Leaf 'a | Node ('a elem × 'a elem)
```

1 (2,3) ((4,5),(6,7)) ((4,5),6)

$$\frac{}{\texttt{full 0 (Leaf x)}} \qquad \frac{\texttt{full n l} \qquad \texttt{full n r}}{\texttt{full (n + 1) (Node (l, r))}}$$

2 overapproximate the set of all powerlists

```
'a plist₀ = PNil₀ | PCons₀ ('a elem) ('a plist₀)
```

3

4

```
'a plist = PNil | PCons 'a (('a × 'a) plist)
```

1 — (2,3) — ((4,5),(6,7))

1 overapproximate the elements of a powerlist

```
'a elem = Leaf 'a | Node ('a elem × 'a elem)
```

1 (2,3) ((4,5),(6,7)) ((4,5),6)

$$\frac{}{\texttt{full 0 (Leaf x)}} \qquad \frac{\texttt{full n l} \qquad \texttt{full n r}}{\texttt{full (n + 1) (Node (l, r))}}$$

2 overapproximate the set of all powerlists

```
'a plist₀ = PNil₀ | PCons₀ ('a elem) ('a plist₀)
```

1 — (2,3) — ((4,5),(6,7)) ((4,5),(6,7)) — (2,3)

3

4

```
'a plist = PNil | PCons 'a (('a × 'a) plist)
```

1 — (2,3) — ((4,5),(6,7))

1 overapproximate the elements of a powerlist

```
'a elem = Leaf 'a | Node ('a elem × 'a elem)
```

1 (2,3) ((4,5),(6,7)) ((4,5),6)

$$\frac{}{\texttt{full 0 (Leaf x)}} \qquad \frac{\texttt{full n l} \qquad \texttt{full n r}}{\texttt{full (n + 1) (Node (l, r))}}$$

2 overapproximate the set of all powerlists

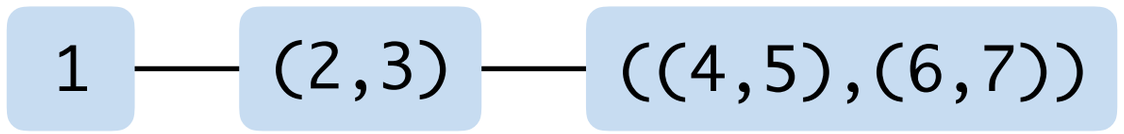
```
'a plist₀ = PNil₀ | PCons₀ ('a elem) ('a plist₀)
```

1 — (2,3) — ((4,5),(6,7)) ((4,5),(6,7)) — (2,3)

$$\frac{}{\texttt{ok n PNil}_0} \qquad \frac{\texttt{full n x} \quad \texttt{ok (n + 1) xs}}{\texttt{ok n (PCons}_0\texttt{ x xs)}}$$

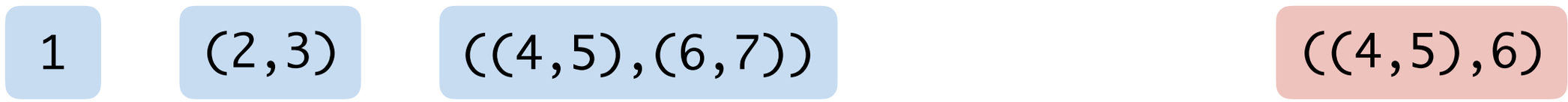
3 4

```
'a plist = PNil | PCons 'a (('a × 'a) plist)
```



1 overapproximate the elements of a powerlist

```
'a elem = Leaf 'a | Node ('a elem × 'a elem)
```



$$\frac{}{\texttt{full 0 (Leaf x)}} \qquad \frac{\texttt{full n l} \qquad \texttt{full n r}}{\texttt{full (n + 1) (Node (l, r))}}$$

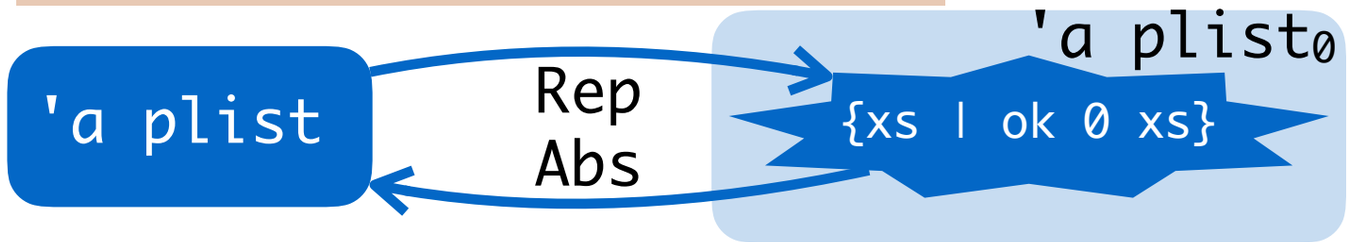
2 overapproximate the set of all powerlists

```
'a plist₀ = PNil₀ | PCons₀ ('a elem) ('a plist₀)
```

$$\frac{}{\texttt{ok n PNil}_0} \qquad \frac{\texttt{full n x} \quad \texttt{ok (n + 1) xs}}{\texttt{ok n (PCons}_0\ \texttt{x xs)}}$$

3 carve out ‘ok’ powerlists

4



```
'a plist = PNil | PCons 'a (('a × 'a) plist)
```

1 —— (2,3) —— ((4,5),(6,7))

1 overapproximate the elements of a powerlist

```
'a elem = Leaf 'a | Node ('a elem × 'a elem)
```

1 (2,3) ((4,5),(6,7)) ((4,5),6)

$$\frac{}{\texttt{full 0 (Leaf x)}} \qquad \frac{\texttt{full n l} \qquad \texttt{full n r}}{\texttt{full (n + 1) (Node (l, r))}}$$

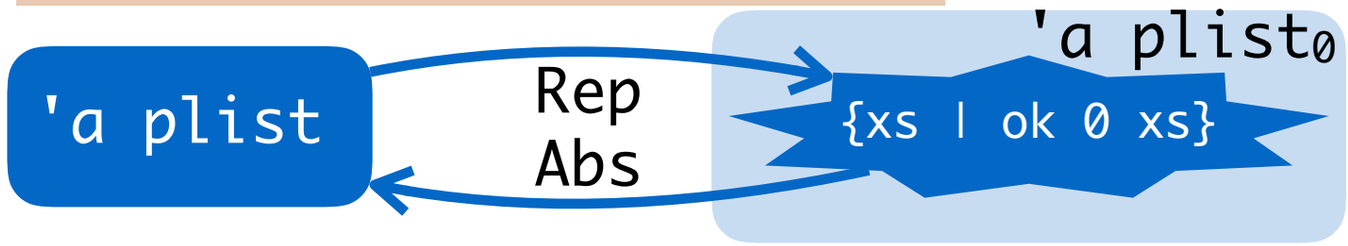
2 overapproximate the set of all powerlists

```
'a plist₀ = PNil₀ | PCons₀ ('a elem) ('a plist₀)
```

1 —— (2,3) —— ((4,5),(6,7)) ((4,5),(6,7)) —— (2,3)

$$\frac{}{\texttt{ok n PNil}_0} \qquad \frac{\texttt{full n x} \quad \texttt{ok (n + 1) xs}}{\texttt{ok n (PCons}_0\ \texttt{x xs)}}$$

3 carve out ‘ok’ powerlists



4 lift constructors

```
PCons x xs = Abs
  (PCons₀ (Leaf x) (… (Rep xs)))
```

General construction supports:

multiple recursive occurrences

```
'a biplist = Nil | Cons₁ 'a (('a list) biplist)
                 | Cons₂ 'a (('a × 'a) biplist)
```

General construction supports:

multiple recursive occurrences

```
'a biplist = Nil | Cons₁ 'a (('a list) biplist)
                 | Cons₂ 'a (('a × 'a) biplist)
```

multiple type arguments

```
('a, 'b) tplist = Nil 'b | Cons 'a (('a × 'a, 'b option) tplist)
```

General construction supports:

multiple recursive occurrences

```
'a biplist = Nil | Cons₁ 'a (('a list) biplist)
                 | Cons₂ 'a (('a × 'a) biplist)
```

multiple type arguments

```
('a, 'b) tplist = Nil 'b | Cons 'a (('a × 'a, 'b option) tplist)
```

mutual definitions

```
'a ptree = Node 'a ('a pforest)
'a pforest = Nil | Cons ('a ptree) (('a × 'a) pforest)
```

General construction supports:

multiple recursive occurrences

```
'a biplist = Nil | Cons₁ 'a (('a list) biplist)
                 | Cons₂ 'a (('a × 'a) biplist)
```

multiple type arguments

```
('a, 'b) tplist = Nil 'b | Cons 'a (('a × 'a, 'b option) tplist)
```

mutual definitions

```
'a ptree = Node 'a ('a pforest)
'a pforest = Nil | Cons ('a ptree) (('a × 'a) pforest)
```

codatatypes

```
'a pstream ≅∞ PSCons 'a (('a list) pstream)
```

General construction supports:

multiple recursive occurrences

```
'a biplist = Nil | Cons₁ 'a (('a list) biplist)
                 | Cons₂ 'a (('a × 'a) biplist)
```

multiple type arguments

```
('a, 'b) tplist = Nil 'b | Cons 'a (('a × 'a, 'b option) tplist)
```

mutual definitions

```
'a ptree = Node 'a ('a pforest)
'a pforest = Nil | Cons ('a ptree) (('a × 'a) pforest)
```

codatatypes

```
'a pstream ≅ PSCons 'a (('a list) pstream)
```

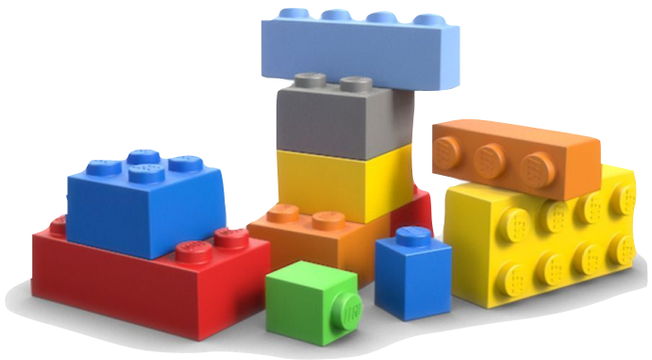
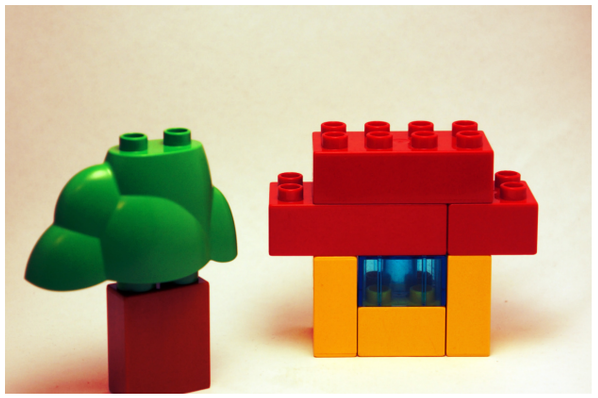
arbitrary bounded natural functors

```
'a crazy = Crazy 'a (((((('a pstream) fset) crazy) multiset) list)
```

Take Home Messages

It is tempting to introduce a new hot logic/language for each new feature.
But this is not always necessary.

The foundational path requires work.


But it also saves work elsewhere.
(keyword: consistency)

Foundational Nonuniform (Co)datatypes for Higher-Order Lo

Bedankt!
Vragen?

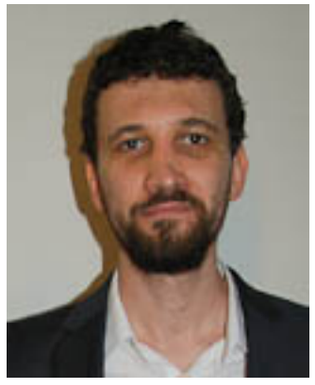
Jasmin Blanchette

VU VRIJE UNIVERSITEIT AMSTERDAM

Fabian Meier

ETH zürich

Andrei Popescu



Middlesex University London

Dmitriy Traytel

Isabelle HOL

ETH zürich