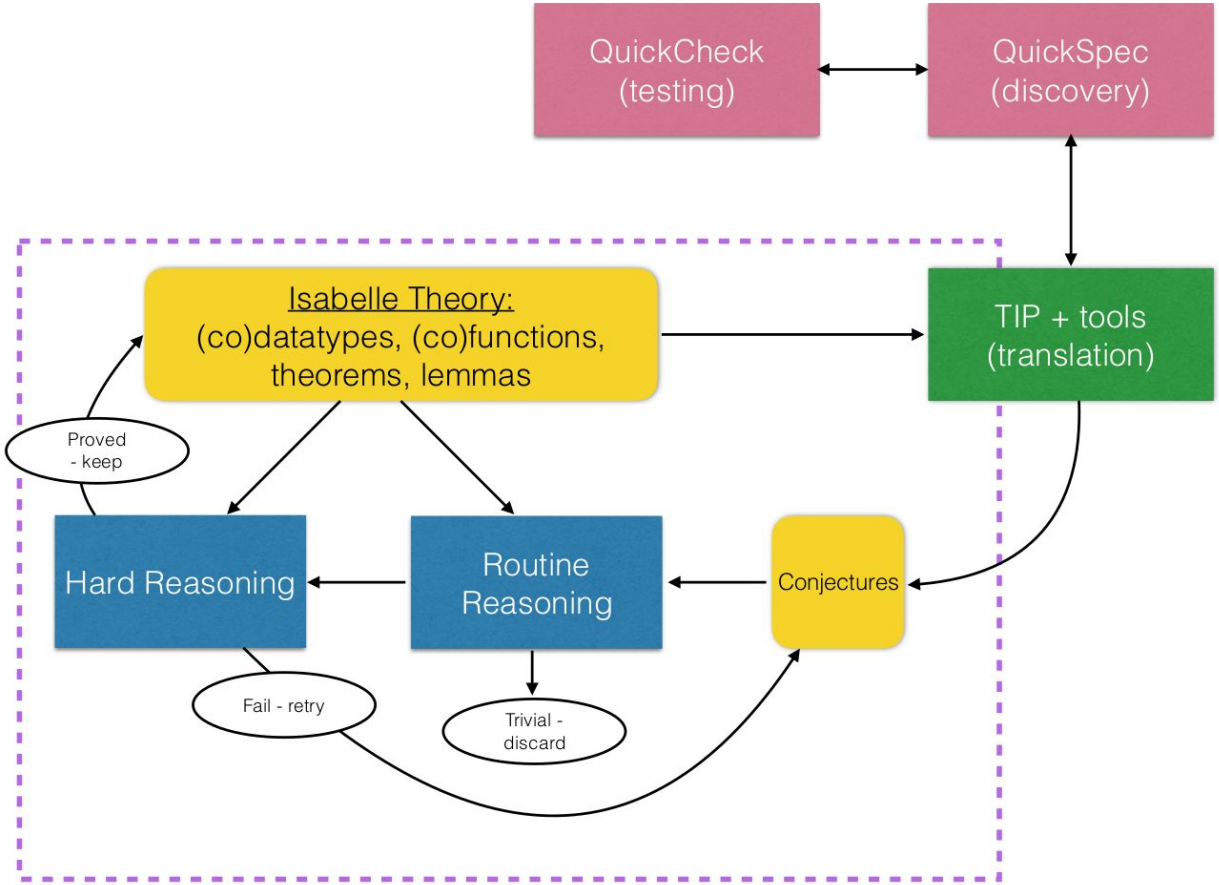
Theory Exploration for Coinduction

Sólrún Halla Einarsdóttir
Moa Johansson Johannes Åman Pohjola

WAIT 2018

DEMO

Hipster


QuickCheck
(testing)
QuickSpec
(discovery)
Isabelle Theory:
(co)datatypes, (co)functions,
theorems, lemmas
TIP + tools
(translation)
Proved
- keep
Hard Reasoning
Routine
Reasoning
Conjectures
Fail - retry
Trivial -
discard

Extending Hipster for coinduction

- Use Isabelle/HOL's support for codatatypes and coinduction
- Extend discovery subsystem to discover properties for infinite codatatypes
- Define automated coinductive proof techniques to prove the discovered lemmas

Discovering properties

Hipster uses QuickSpec to discover equational properties:

- Generate terms
- Instantiate terms with arbitrary values
- Compare for equality

What happens when the terms are infinitely large?

Testing infinite structures

- How do we compare infinite structures for equality?
- Observational equivalence
  - Observe finite substructures and compare them instead

Observer function

To observe a type T the user may provide an observer function with type

```
Obs -> T -> Res
```

`Obs`: Type we can generate arbitrary instances for

`Res`: Type that can be compared for equality

```
obsStream :: Nat -> Stream -> List
```

Automatically generating observers

- Observer type, same as original type + additional nullary constructor
- Observer function has a decrementing counter, replaces constructors with equivalent observer type constructors
- Example: Streams

```
Stream a = SCons a (Stream a)

OStream a = OSCons a (OStream a) | NCStream
```

Automatically generating observers

```
obsStream :: Nat -> Stream a -> OStream a

obsStream 0 _              = NCStream

obsStream n (SCons x xs) = OSCons x

                             (obsStream (n-1) xs)
```

Automatically generating observers

- Decrementing counter *n* to *n-1* too inefficient for non-linear branching
- Divide by number of recursive constructors (i.e. n/2 - 1 for binary tree)
- Some support for observing nested codatatypes by generating helper observers

Automated coinductive proofs

- Use Isabelle/HOL’s built-in coinduction tactic
- Automatically determine parameters for call to coinduction tactic
- Automate subgoal proofs

Determining arbitrary variables

- *Arbitrary* variables (universally quantified in candidate relation - existentially quantified in subgoal proof)
- Set all free variables in goal as arbitrary
- Weaker proof obligations, at the expense of introducing existential quantifiers in the goal
- Works well in practice

Determining coinduction rule

- Strong coinduction principle generated by codatatype
- Candidate relation instantiated to singleton relation containing the equation in the conclusion
- Works well for primitively corecursive functions

$$\frac{R\ s\ s' \qquad \forall s_1, s_2\ \dfrac{R\ s_1\ s_2}{shd\ s_1 = shd\ s_2 \wedge (R\ (stl\ s_1)\ (stl\ s_2) \vee\ stl\ s_1 = stl\ s_2))}}{s = s'}$$

Proving subgoals

- First, attempt to prove subgoals using Isabelle/HOL's *simp* tactic
- If that doesn't work, call Sledgehammer
- Works well in practice

Mixed induction/coinduction

- Many coinductive definitions depend on auxiliary inductive definitions and vice versa
- Heuristic:
  - If conjecture contains a free variable whose type has an induction principle, try induction
  - If LHS and RHS of conjecture are of a type that has a coinduction principle, try coinduction
  - If both, try both and keep first successful proof

Evaluation

- Do we discover interesting or useful lemmas?
- We think so! (as shown in Demo earlier)
- Coinductive benchmark suite?

Hinze’s stream calculus laws

- “provides an account of the most important properties of streams”
- We discover 9 out of 18
- 3 out of scope
- QuickSpec’s heuristics for restricting search space

	Property	Found	Precision	Recall	Time
1	$pure\ id \diamond u = u$	X			
2	$pure(\circ) \diamond u \diamond v \diamond w \diamond u = u$	–	22%	67%	44s.
3	$pure\ f \diamond pure\ x = pure\ (f\ x)$	X			
4	$u \diamond pure\ x = pure\ (\lambda f.\ f\ x) \diamond u$				
5	$map\ id\ x = x$	X	50%	100%	29s.
6	$map\ (f \circ g)\ x = map\ f\ (map\ g\ x)$	X			
7	$map\ fst\ (zip\ s\ t) = s$	–			
8	$map\ snd\ (zip\ s\ t) = t$	–	0%	0%	255s.
9	$zip\ (map\ fst\ p)\ (map\ snd\ p) = p$	–			
10	$pure\ a \curlyvee pure\ a = pure\ a$	X	25%	50%	18s.
11	$(s_1 \diamond s_2) \curlyvee (t_1 \diamond t_2) = (s_1 \curlyvee t_1) \diamond (s_2 \curlyvee t_2)$	–			
12	$map\ f\ (tabulate\ g) = tabulate\ (f \circ g)$	X	100%	100%	87s.
13	$f(lookup\ t\ x) = lookup\ (map\ f\ t)\ x$	X	33%	100%	57s.
14	$recurse\ f\ a = iterate\ f\ a$	X	33%	100%	73s.
15	$map\ h \circ iterate\ f_1 = iterate\ f_2 \circ h \Longleftarrow h \circ f_1 = f_2 \circ h$				
16	$unfold\ hd\ tl\ x = x$	–	0%	0%	21s.
17	$unfold\ g\ f \circ h = unfold\ g'\ f' \Longleftarrow g \circ h = g' \wedge f \circ h = h \circ f'$				
18	$map\ h\ (unfold\ g\ f\ x) = unfold\ (h \circ g)\ f\ x$	X	50%	100%	18s.
			21%	**60%**	**602s.**

Coinductive library in AFP

- https://www.isa-afp.org/entries/Coinductive.html
- Very large theory files - difficult to test systematically
- Many lemmas out of scope for us (lambdas, no code equations…)
- But we do find many lemmas which are stated and proved there!

Conclusions & future work

- First theory exploration system to handle coinductive lemma discovery!
- Future work
  - Better specialized observer functions and proof methods to better handle i.e. nested types
  - Extend to other uses of coinduction like reasoning about behavioral equivalences or non-terminating programs

Thanks!

- Hipster: https://github.com/moajohansson/IsaHipster
- Paper: http://www.cse.chalmers.se/~slrn/papers/into_infinite.pdf