# HipSpec

## Automating Inductive Proofs using Theory Exploration

Dan Rosén

Koen Claessen, Moa Johansson, Nicholas Smallbone

Chalmers University of Technology

May 31, 2013

## Rotate example

```
rotate :: Nat -> [a] -> [a]
rotate Z     xs     = xs
rotate (S n) []     = []
rotate (S n) (x:xs) = rotate n (xs ++ [x])
```

```
rotate 1 [1,2,3,4] = [2,3,4,1]
rotate 2 [1,2,3,4] = [3,4,1,2]
rotate 3 [1,2,3,4] = [4,1,2,3]
rotate 4 [1,2,3,4] = [1,2,3,4]
```

## Rotate example

```
rotate :: Nat -> [a] -> [a]
rotate Z     xs     = xs
rotate (S n) []     = []
rotate (S n) (x:xs) = rotate n (xs ++ [x])
```

```
rotate 1 [1,2,3,4] = [2,3,4,1]
rotate 2 [1,2,3,4] = [3,4,1,2]
rotate 3 [1,2,3,4] = [4,1,2,3]
rotate 4 [1,2,3,4] = [1,2,3,4]
```

$$\forall \texttt{xs}.\, \texttt{rotate (length xs) xs} = \texttt{xs}$$

## Rotate example

```
rotate :: Nat -> [a] -> [a]
rotate Z     xs     = xs
rotate (S n) []     = []
rotate (S n) (x:xs) = rotate n (xs ++ [x])
```

$$\forall\, \texttt{xs}.\, \texttt{rotate (length xs) xs} = \texttt{xs}$$

## Rotate example

```
rotate :: Nat -> [a] -> [a]
rotate Z     xs     = xs
rotate (S n) []     = []
rotate (S n) (x:xs) = rotate n (xs ++ [x])
```

$$\forall \mathtt{xs}. \mathtt{rotate\ (length\ xs)\ xs} = \mathtt{xs}$$

$$\begin{aligned}
&\mathtt{rotate\ (length\ (a{:}as))\ (a{:}as)} &=\\
&\mathtt{rotate\ (S\ (length\ as))\ (a{:}as)} &=\\
&\mathtt{rotate\ (length\ as)\ (as\ {+}{+}\ [a])} &=
\end{aligned}$$

## Rotate example

```
rotate :: Nat -> [a] -> [a]
rotate Z     xs     = xs
rotate (S n) []     = []
rotate (S n) (x:xs) = rotate n (xs ++ [x])
```

$$\forall \texttt{xs}.\texttt{rotate (length xs) xs} = \texttt{xs}$$

$$\begin{aligned}
&\texttt{rotate (length (a:as)) (a:as)} &=\\
&\texttt{rotate (S (length as)) (a:as)} &=\\
&\texttt{rotate (length as) (as ++ [a])} &=
\end{aligned}$$

Stuck!

## HipSpec vs Rotate

$$\forall\, \texttt{xs}, \texttt{ys}.\, \texttt{rotate (length xs) (xs ++ ys)} = \texttt{ys ++ xs}$$

## HipSpec vs Rotate

$$\forall\, \texttt{xs}, \texttt{ys}.\, \texttt{rotate (length xs) (xs ++ ys)} = \texttt{ys ++ xs}$$

(also requires associativity and right identity of ++)

## QuickSpec: the Theory Exploration Phase

Generates well-typed terms up to some depth:

```
rot (len xs) xs    len xs              xs++(ys++ys)
rot n (xs++xs)     rot n (rot m xs)    rot n xs++rot n xs
(xs++ys)++ys       rot Z (xs++ys)      rot m (rot n xs)
xs                 len (rot m xs)      len (rot n xs)
xs++ys             len (ys++xs)        len (rot o xs)
rot Z xs           len (xs++ys)        []++xs
(xs++ys)++[]       xs++[]              rot (len ys) (ys++xs)
```

# Partitioning into Equivalence Classes

```
xs
xs++[]
[]++xs
rot Z xs
rot (len xs) xs
```

```
xs++ys
[]++(xs++ys)
rot Z (xs++ys)
(xs++ys)++[]
rot (len ys) (ys++xs)
```

```
xs++(ys++ys)
(xs++ys)++ys
```

```
rot n (xs++xs)
rot n xs++rot n xs
```

```
len (xs++ys)
len (ys++xs)
```

```
len xs
len (rot n xs)
```

```
rot n (rot m xs)
rot m (rot n xs)
```

## Hip: The Haskell Inductive Prover

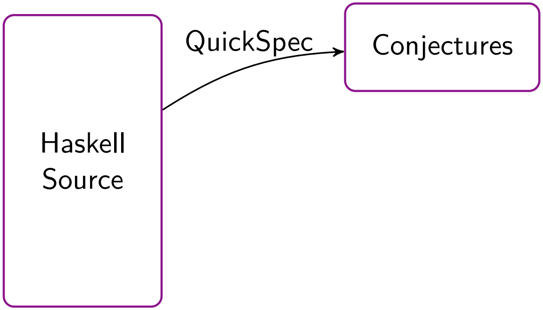
- Translate to typed first order logic
- Apply structural induction

Also supports higher-order functions and partial application

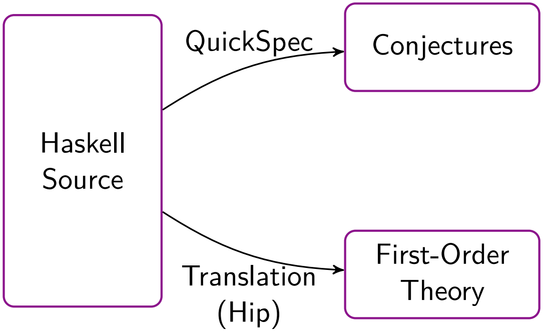
## Overview of HipSpec

Haskell
Source

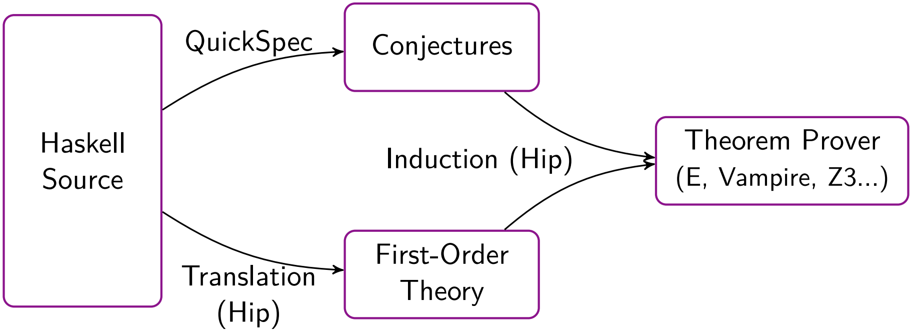
## Overview of HipSpec



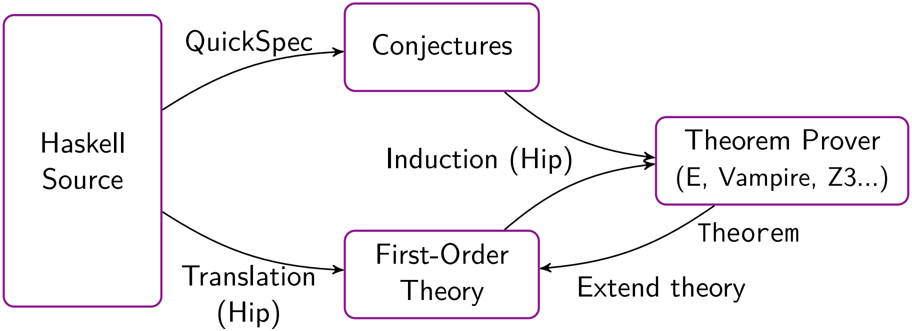
## Overview of HipSpec



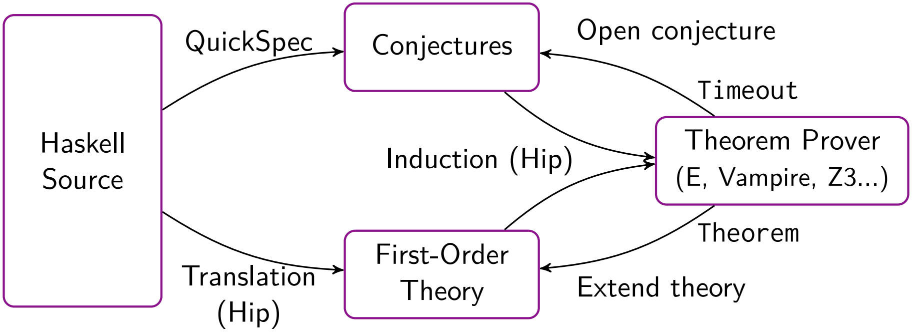
## Overview of HipSpec



# Overview of HipSpec



## Overview of HipSpec



## Prioritising Equations

1. Call graph

## Prioritising Equations

1. Call graph

```
xs++[] = xs
length (xs++ys) = length (ys++xs)
rotate (length xs) (xs ++ ys) = ys ++ xs
```

## Prioritising Equations

1. Call graph
2. Size of term

```
xs++[] = xs
(xs++ys)++zs = xs++(ys++zs)
```

## Prioritising Equations

1. Call graph
2. Size of term
3. Number of variables

```
(xs++ys)++zs = xs++(ys++zs)
(xs++xs)++ys = xs++(xs++ys)
(xs++xs)++xs = xs++(xs++xs)
```

## Evaluation Results

1st test suite from *Case-analysis for Rippling and Inductive Proof*:

| #Props | HipSpec | Zeno | ACL2s | IsaPlanner | Dafny |
|---|---|---|---|---|---|
| 85 | 80 | 82 | 74 | 47 | 45 |

## Evaluation Results

1st test suite from *Case-analysis for Rippling and Inductive Proof*:

| #Props | HipSpec | Zeno | ACL2s | IsaPlanner | Dafny |
|---|---|---|---|---|---|
| 85 | 80 | 82 | 74 | 47 | 45 |

2nd test suite from *Productive use of Failure in Inductive Proof*:

| #Props | HipSpec | CLAM | Zeno |
|---|---|---|---|
| 50 | 44 | 41 | 21 |

## Conjecturing Conditionals

$$\forall\, \texttt{xs}.\, \texttt{sorted (isort xs)} = \texttt{True}$$

```
isort :: [Nat] -> [Nat]

insert :: Nat -> [Nat] -> [Nat]

sorted :: [Nat] -> Bool
```

## Conjecturing Conditionals

$$\forall\, \texttt{xs}.\, \texttt{sorted (isort xs)} = \texttt{True}$$

```
isort :: [Nat] -> [Nat]

insert :: Nat -> [Nat] -> [Nat]

sorted :: [Nat] -> Bool
```

Requires:

$$\forall\, \texttt{xs}.\, \texttt{sorted xs} = \texttt{True} \Rightarrow \texttt{sorted (insert x xs)} = \texttt{True}$$

## $\top$ vs $\bot$

- Top-down: Rippling/critics-based provers, ACL, Zeno

## $\top$ vs $\bot$

- Top-down: Rippling/critics-based provers, ACL, Zeno

$$\forall\, \texttt{xs}.\, \texttt{rotate (length xs) xs} = \texttt{xs}$$

$$\begin{aligned}
\texttt{rotate (length (a:as)) (a:as)} &= \\
\texttt{rotate (S (length as)) (a:as)} &= \\
\texttt{rotate (length as) (as ++ [a])} &=
\end{aligned}$$

Stuck!

## $\top$ vs $\bot$

- Top-down: Rippling/critics-based provers, ACL, Zeno

$$\forall\, \mathtt{i}, \mathtt{xs}.\, \mathtt{rev\ (drop\ i\ xs)} = \mathtt{take\ (length\ xs\ -\ i)\ (rev\ xs)}$$

## $\top$ vs $\bot$

- Top-down: Rippling/critics-based provers, ACL, Zeno

$\forall$ `i, xs. rev (drop i xs)` $=$ `take (length xs - i) (rev xs)`

Required lemmas:

```
length (drop x xs)            = length xs - x
length (rev xs)               = length xs
take x xs ++ drop x xs        = xs
rev xs ++ rev ys              = rev (ys++xs)
take (length xs) (xs ++ ys) = xs
```

## $\top$ vs $\bot$

- Top-down: Rippling/critics-based provers, ACL, Zeno
- Bottom-up: IsaCoSy, IsaScheme, HipSpec

$\forall$ `i, xs. rev (drop i xs)` $=$ `take (length xs - i) (rev xs)`

Required lemmas:

```
length (drop x xs)             = length xs - x
length (rev xs)                = length xs
take x xs ++ drop x xs         = xs
rev xs ++ rev ys               = rev (ys++xs)
take (length xs) (xs ++ ys) = xs
```

## HipSpec the Theory Exploration System

Precision/recall analysis against Isabelle standard library

## HipSpec the Theory Exploration System

Precision/recall analysis against Isabelle standard library

What the other systems do in hours, HipSpec does *under a minute*!

## HipSpec the Theory Exploration System

Precision/recall analysis against Isabelle standard library

What the other systems do in hours, HipSpec does *under a minute*!

- `data BinNat = Zero | ZeroAnd BinNat | OneAnd BinNat`

## HipSpec the Theory Exploration System

Precision/recall analysis against Isabelle standard library

What the other systems do in hours, HipSpec does *under a minute*!

- `data BinNat = Zero | ZeroAnd BinNat | OneAnd BinNat`
- `data Integer = Positive Nat | Negative Nat`

## Conclusions

- Evaluate your programs!
- Completeness up to a certain depth:
  *If the lemma is there, HipSpec will eventually try to prove it!*

github.com/danr/hipspec

## Conditionals as functions

$$\forall\, \texttt{xs}.\ \texttt{sorted (isort xs)} = \texttt{True}$$

```
whenSorted :: [Nat] -> [Nat]
whenSorted xs = if sorted xs then xs else []
```

$$\forall\, \texttt{x}, \texttt{xs}.\ \texttt{sorted (insert x (whenSorted xs))} = \texttt{True}$$

```
  sorted (insert x (whenSorted xs))
= sorted (insert x (if sorted xs then xs else []))
= if sorted xs then sorted (insert x xs)
               else sorted (insert x [])
```

## What is HipSpec?

**Haskell source**

```
rev [] = []
rev (x:xs)
  = rev xs ++ [x]

prop_rev xs
  = rev (rev xs) =:= xs
```

**Hip**

*Haskell Inductive Prover*

- FOL translation
- Apply induction
- Success, or stuck!

**QuickSpec**

Eq-theory from testing:

```
rev (xs ++ ys)
  = rev ys ++ rev xs
xs ++ [] = []
xs ++ (ys ++ zs) =
  (xs ++ ys) ++ zs
```

**HipSpec**

*Use these as lemmas!!*