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])
```

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

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

Rotate example

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

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

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

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
qrev [] xs
rev (rev xs)
qrev (rev xs) []
```

```
[]
rev []
qrev [] []
[]++[]
```

```
qrev xs ys
rev (qrev ys xs)
rev xs++ys
[]++qrev xs ys
qrev [] (qrev xs ys)
qrev xs ys++[]
qrev (qrev ys xs) []
```

```
xs++ys
qrev (rev xs) ys
[]++(xs++ys)
qrev [] (xs++ys)
(xs++ys)++[]
```

```
x:xs
[]++(x:xs)
qrev [] (x:xs)
(x:xs)++[]
(x:[])++xs
qrev (x:[]) xs
```

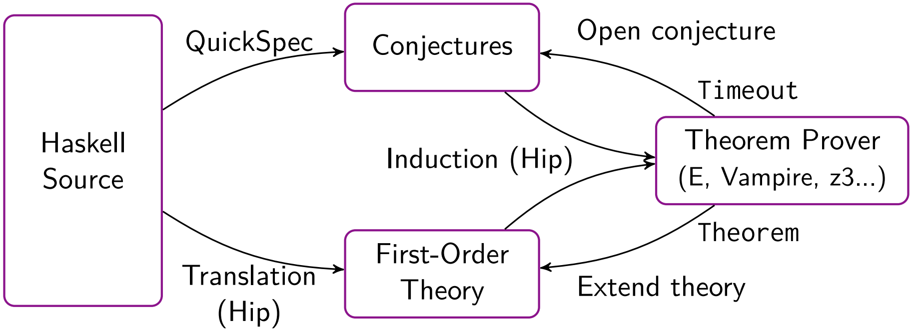
```
rev xs
qrev xs []
[]++rev xs
qrev [] (rev xs)
```

Hip: The Haskell Inductive Prover

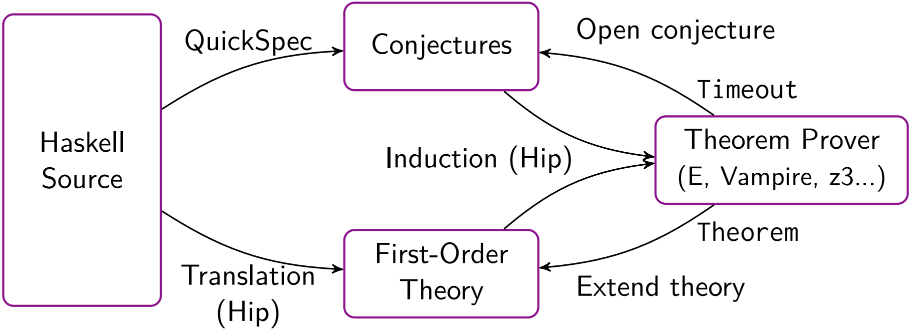
- Translate to typed first order logic
- Apply structural induction

Also supports higher-order functions and partial application

Overview of HipSpec



Overview of HipSpec



- Try to prove "smallest" unproved equation
- Terminate everything is proved (or when the current theory cannot prove any more open conjectures)

Prioritising Equations

- Call graph
- Number of variables
- Size of term

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

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

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

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

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

Conjecturing Conditionals

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

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

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

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

Requires:

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

Example tricky equational proof

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

Example tricky equational proof

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

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
```

Rotate, revisited

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

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

Stuck!

Rotate, revisited

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

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

Stuck!

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

Rotate, revisited

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

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

Stuck!

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

HipSpec the Theory Exploration System

Saturate a theory and have it nicely presented

HipSpec the Theory Exploration System

Saturate a theory and have it nicely presented

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

Theory Exploration Results

	Isabelle
#Thms Nats	12
Precision	-
Recall	-
#Thms Lists	9
Precision	-
Recall	-

Theory Exploration Results

	Isabelle	HipSpec
#Thms Nats	12	10
Precision	-	80%
Recall	-	73%
#Thms Lists	9	
Precision	-	
Recall	-	

Theory Exploration Results

	Isabelle	HipSpec	IsaCoSy	IsaScheme
#Thms Nats	12	10	16	16*
Precision	-	80%	63%	100%*
Recall	-	73%	83%	46%*
#Thms Lists	9			
Precision	-			
Recall	-			

Theory Exploration Results

	Isabelle	HipSpec	IsaCoSy	IsaScheme
#Thms Nats	12	10	16	16*
Precision	-	80%	63%	100%*
Recall	-	73%	83%	46%*
#Thms Lists	9	10	24	13
Precision	-	90%	38%	70%
Recall	-	100%	100%	100%

Theory Exploration Results

	Isabelle	HipSpec	IsaCoSy	IsaScheme
#Thms Nats	12	10	16	16*
Precision	-	80%	63%	100%*
Recall	-	73%	83%	46%*
#Thms Lists	9	10	24	13
Precision	-	90%	38%	70%
Recall	-	100%	100%	100%
Runtime		30 seconds	hours	hours

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

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

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

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

```
  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!!