HipSpec

Automating Inductive Proofs using Theory Exploration

Dan Rosén

Koen Claessen, Moa Johansson, Nicholas Smallbone

Chalmers University of Technology

Rotate Example

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

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

```
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 Zero     xs     = xs
rotate (Succ n) []     = []
rotate (Succ n) (x:xs) = rotate n (xs ++ [x])
```

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

hypothesis:

$$\texttt{rotate (length as) as} = \texttt{as}$$

conclusion:

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

Rotate Example

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

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

hypothesis:

$$\texttt{rotate (length as) as} = \texttt{as}$$

conclusion:

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

Stuck!

Rotate-length Helper Lemma

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

Rotate-length Helper Lemma

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

conclusion:

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

hypothesis:

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

Rotate-length Helper Lemma

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

conclusion:

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

hypothesis:

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

Bundy, Basin, Hutter, Ireland: automated induction challenge in *Rippling: meta-level guidance for mathematical reasoning*

QuickSpec: the Theory Exploration Phase

Generates well-typed terms up to some depth:

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

Partitioning into Equivalence Classes

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

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

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

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

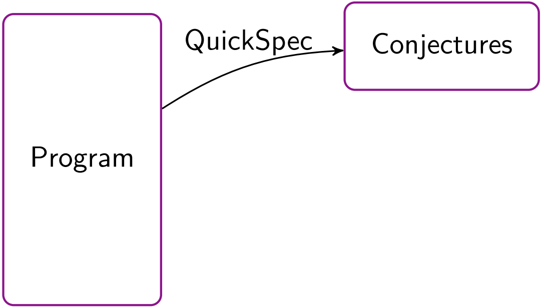
- Translate to typed first order logic
- Apply structural induction

Also supports higher-order functions and partial application

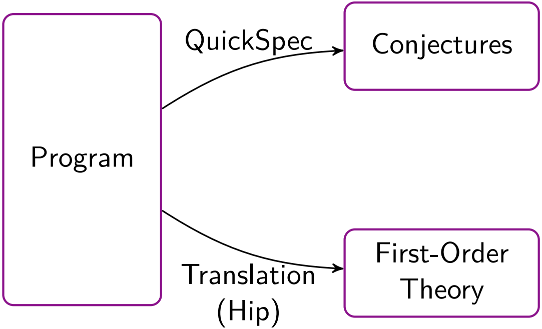
Overview of HipSpec

Program

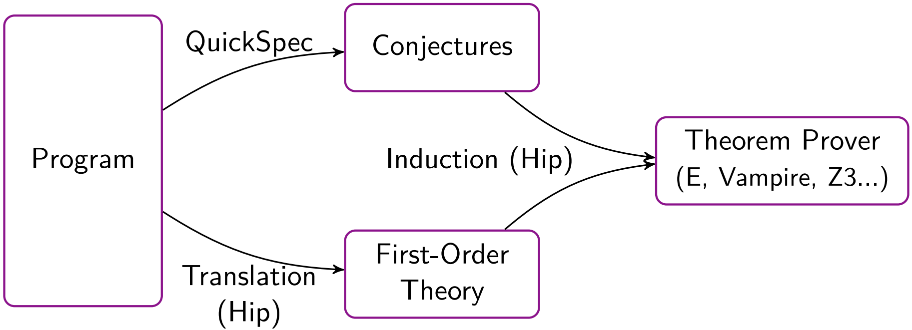
Overview of HipSpec



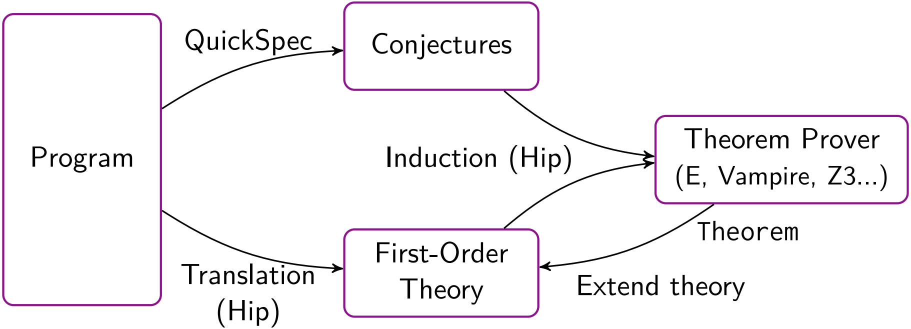
Overview of HipSpec



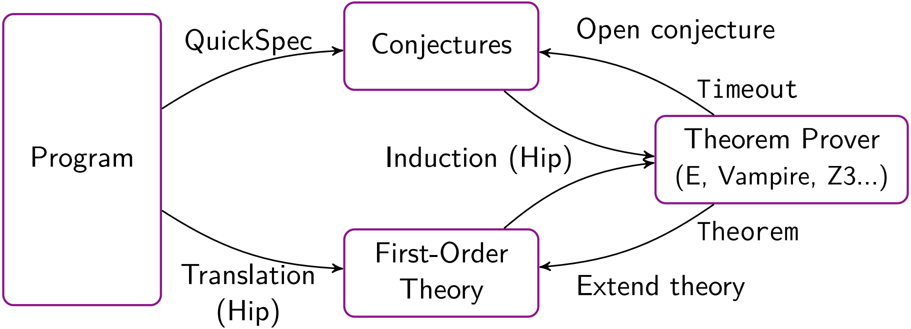
Overview of HipSpec



Overview of HipSpec

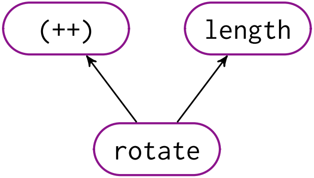


Overview of HipSpec



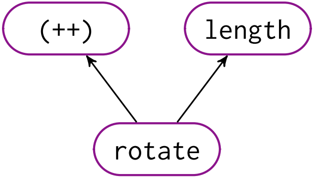
Prioritising Equations

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 I

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

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

- Quite easy: around 60 provable without lemmas
- Some require conditional lemmas
  (can't generate with QuickSpec)

Evaluation Results II

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

#Props	HipSpec	CLAM	Zeno
50	44	41	21

- Harder test suite: need lemmas and generalisations

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:

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

Two Approaches to Lemma Discovery

- Top-down: Rippling/critics-based provers, ACL2, Zeno

Two Approaches to Lemma Discovery

- Top-down: Rippling/critics-based provers, ACL2, 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!

Two Approaches to Lemma Discovery

- Top-down: Rippling/critics-based provers, ACL2, Zeno

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

Two Approaches to Lemma Discovery

- Top-down: Rippling/critics-based provers, ACL2, Zeno

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

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
```

Two Approaches to Lemma Discovery

- Top-down: Rippling/critics-based provers, ACL2, 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
```

Theory Exploration Results

- Produce background theory comparable to human.
- Comparison with Isabelle libraries.
- **HipSpec runs in minutes.** IsaCoSy/IsaScheme sometimes hours.

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

Table : Theory Exploration results. IsaScheme was evaluated on a natural number theory also including exponentiation.

- `data Integer = Positive Nat | Negative Nat`

Conclusions

- Evaluate your programs

Conclusions

- Evaluate your programs
- "Completeness" up to a certain depth

Conclusions

- Evaluate your programs
- "Completeness" up to a certain depth
- Progress in automated induction

github.com/danr/hipspec

Conditionals as Functions

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

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

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

Conditionals as Functions

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

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

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

```
  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!!