Hipster: Integrating Theory Exploration in a Proof Assistant

Moa Johansson
Joint work with Dan Rosén, Nick Smallbone and Koen Claessen
Chalmers University, Gothenburg, Sweden.

Conference on Intelligent Computer Mathematics
Coimbra, Portugal
9 July 2014

Introduction: Theory Exploration

Theory Exploration Paradigm [Buchberger-2000]:

- Theorems not proved in isolation.
- Rather, explore whole theories:
  - Prove routine lemmas.
  - Proceed to more complex theorems.
  - Possibly backtrack and prove more lemmas.
  - New theories on top of old ones.
- Interactive theorem proving:
  - Creative/hard steps left to user.

Introduction: Theory Exploration

Theory Exploration Paradigm [Buchberger-2000]:

- Theorems not proved in isolation.
- Rather, explore whole theories:
  - Prove routine lemmas.
  - Proceed to more complex theorems.
  - Possibly backtrack and prove more lemmas.
  - New theories on top of old ones.
- Interactive theorem proving:
  - Creative/hard steps left to user.

Our work:

Automatically discover **new** and **interesting** lemmas in inductive theories.

Inductive Theorem Proving and Theory Exploration

Example Domain: Proofs by induction

- Often need lemmas (also needing induction).
- Hard to find automatically, e.g. generalisations.
- **Bottom-up approach:** Create richer background theory first.

Inductive Theorem Proving and Theory Exploration

Example Domain: Proofs by induction

- Often need lemmas (also needing induction).
- Hard to find automatically, e.g. generalisations.
- **Bottom-up approach:** Create richer background theory first.

Background: HipSpec

- Inductive prover for Haskell.
- Generate (equational) conjectures. Tested, not proved.
- Apply induction, then call off the shelf FO-provers.

Hipster: Theory Exploration for Isabelle/HOL

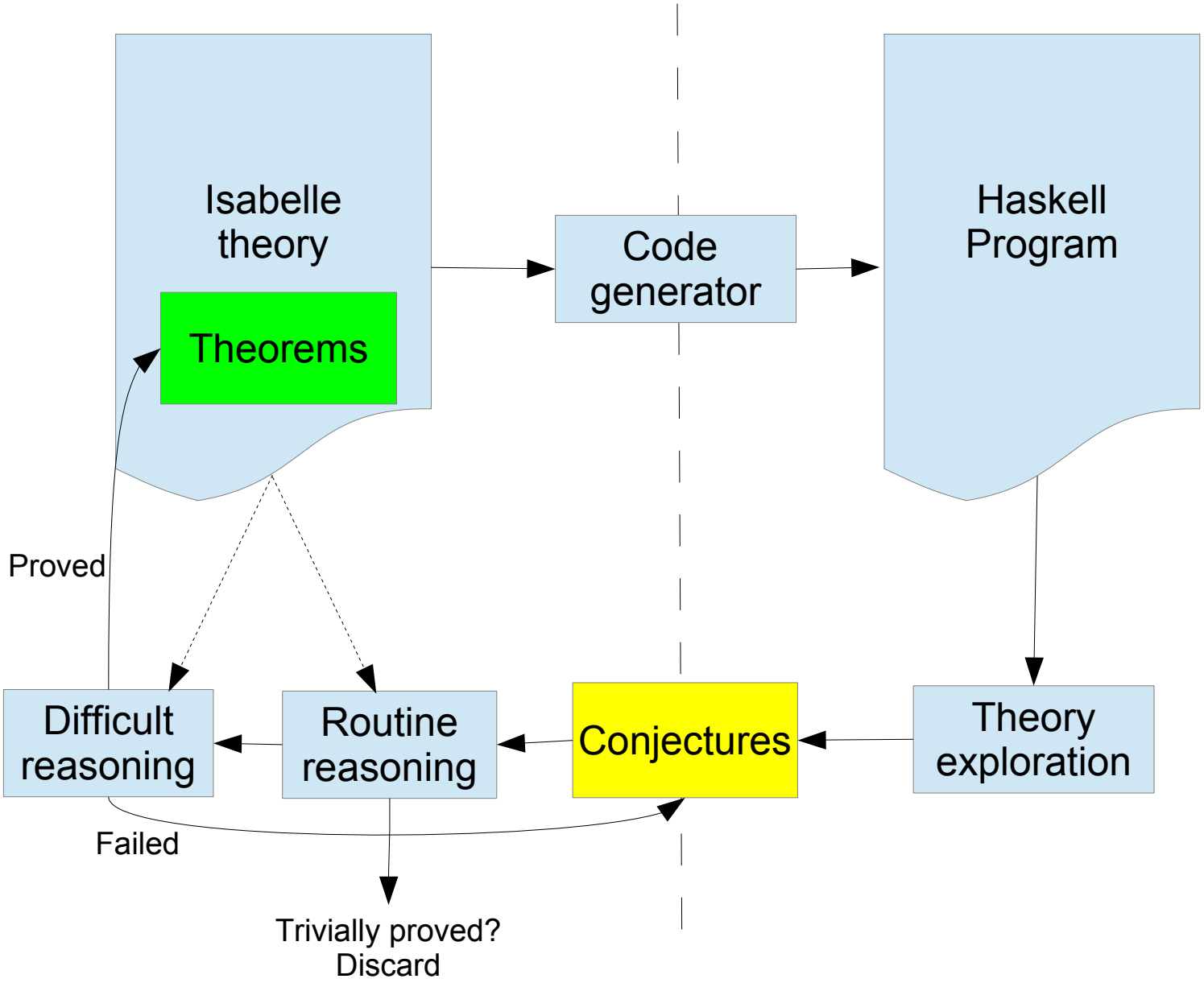
- Translate Isabelle/HOL theory to Haskell.
- Use conjecture generation from HipSpec.
- Currently only equational conjectures.
- Prove in Isabelle (LCF-style).
  - Keep interesting theorems (need induction).
  - Discard if trivial proof.

Hipster: Theory Exploration for Isabelle/HOL

- Translate Isabelle/HOL theory to Haskell.
- Use conjecture generation from HipSpec.
- Currently only equational conjectures.
- Prove in Isabelle (LCF-style).
  - Keep interesting theorems (need induction).
  - Discard if trivial proof.

Demo: Exploring a theory about binary trees

Hipster: Overview



Conjecture Generation in Haskell

- Set of *functions* and *variables*.
- All type-correct terms up to given depth.
- Testing (many) random ground instances.
- Evaluate and divide equivalence classes.

Conjecture Generation in Haskell

- Set of *functions* and *variables*.
- All type-correct terms up to given depth.
- Testing (many) random ground instances.
- Evaluate and divide equivalence classes.

Example: `xs` $\mapsto$`[]`, `ys` $\mapsto$`[a]`, `zs` $\mapsto$`[b]`

Term	Ground Instance	Value
`(xs @ ys) @ zs`		
`xs @ (ys @ zs)`		
`xs @ []`		
`xs`		

Conjecture Generation in Haskell

- Set of *functions* and *variables*.
- All type-correct terms up to given depth.
- Testing (many) random ground instances.
- Evaluate and divide into equivalence classes.

Example: `xs` $\mapsto$`[]`, `ys` $\mapsto$`[a]`, `zs` $\mapsto$`[b]`

Term	Ground Instance	Value
`(xs @ ys) @ zs`	`([] @ [a]) @ [b]`	
`xs @ (ys @ zs)`	`[] @ ([a] @ [b])`	
`xs @ []`	`[] @ []`	
`xs`	`[]`	

Conjecture Generation in Haskell

- Set of *functions* and *variables*.
- All type-correct terms up to given depth.
- Testing (many) random ground instances.
- Evaluate and divide into equivalence classes.

Example: `xs` $\mapsto$`[]`, `ys` $\mapsto$`[a]`, `zs` $\mapsto$`[b]`

Term	Ground Instance	Value
`(xs @ ys) @ zs`	`([] @ [a]) @ [b]`	`[a,b]`
`xs @ (ys @ zs)`	`[] @ ([a] @ [b])`	`[a,b]`
`xs @ []`	`[] @ []`	`[]`
`xs`	`[]`	`[]`

Ongoing and Further Work

- Experiments with different tactics for hard/routine reasoning.
- **Conditional lemmas:**
  - Given a side condition, generate lemmas.
  - E.g. `sorted(xs) ==> sorted(insert x xs)`

Conclusion

- Automatically find and prove routine lemmas.
- LCF-style re-checkable proofs.
- Incremental exploration, store lemmas in libraries.
- User can control search space.
- Lemmas enhance automated tactics, e.g. Sledgehammer.