A TALE of TWO PROVERS

by Niki Vazou, Leonidas Lampropoulos and Jeff Polakow
Haskell

take :: Int → [a] → [a]
Liquid Haskell

\[
\text{take} :: \text{i : } \{ \text{Int} \mid 0 \leq i \} \to \text{xs : } \{ [a] \mid i \leq \text{len xs} \} \to [a]
\]
take :: \i: \{ \text{Int} | 0 \leq i \} \rightarrow \text{xs}: \{ [a] | i \leq \text{len} \ \text{xs} \} \rightarrow [a] 

\text{take } 2 \ [1,2,3] \ \text{OK} \\
\text{take } 9 \ [1,2,3] \ \text{Error}
Liquid Haskell

take :: i:{{Int|0≤i}} → xs:{{[a]|i≤len xs}} → [a]

take 2 [1,2,3] → 0 ≤ 2 ≤ 3 → OK

take 9 [1,2,3] → 0 ≤ 9 ≤ 3 → Error
Is Liquid Haskell a Theorem Prover?
Is **Liquid Haskell** a Theorem Prover?

**Theorem**: *Parallelism Equivalence*

If $f$ is a morphism* between two lists, then $f$ can be applied in parallel.

* $f$ is a morphism when
  
$$ f \; [] = [] \land f \; (x \langle \rangle y) = f \; x \; \langle \rangle \; f \; y $$
Is Liquid Haskell a Theorem Prover?

**Theorem:** Parallelism Equivalence
If $f$ is a morphism* between two lists, then $f \ x = \text{concat} \ (\text{pmap} \ f \ (\text{chunk} \ i \ x))$.

*$f$ is a morphism when

$$f \ [] = [] \land f \ (x<>y) = f \ x \ <> f \ y$$
Is **Liquid Haskell** a Theorem Prover?

```haskell
pEquiv :: f:([a] -> [b])
    -> Morphism [a] [b] f
    -> x:[a] -> i:Pos
    -> {f x = concat (pmap f (chunk i x))}
```

*f* is a morphism when

\[
f \; [] = [] \land f \; (x<>y) = f \; x \; <> \; f \; y
\]
pEquiv :: f:([a] -> [b])
    -> Morphism [a] [b] f
    -> x:([a] -> i:Pos
    -> {f x = concat (pmap f (chunk i x))}

*type Morphism a b f = x:a -> y:b ->
  {f []=[] ∧ f (x<>y) = f x <> f y}
Is **Liquid Haskell** a Theorem Prover?

Yes!

\[
pEquiv :: f :\left(\left[a\right] \rightarrow \left[b\right]\right) \\
\rightarrow \text{Morphism} \ \left[a\right] \left[b\right] f \\
\rightarrow x : \left[a\right] \rightarrow i : \text{Pos} \\
\rightarrow \left\{ f \ x = \text{concat} \ (\text{pmap} \ f \ (\text{chunk} \ i \ x)) \right\}
\]

**Theorems:** Refinement Types

**Proofs:** (Terminating) Haskell Terms

**Correctness:** Liquid Type Checking
Is **Liquid Haskell** a Theorem Prover? Yes!

\[
pEquiv :: f:([a] -> [b]) \\
     -> \text{Morphism} \ [a] [b] f \\
     -> x:[a] -> i:Pos \\
     -> \{ f x = \text{concat} (pmap f (\text{chunk} i x)) \}
\]

Demo
Morphism Parallelism Equivalence

pEquiv :: (Chunkable n, Monoid m) => f:(n -> m) -> Morphism n m f -> x:n -> i:Pos -> \{ f x = mconcat (pmap f (chunk i x)) \}
**Application:** String Matching

Find all the occurrences of a *target* string in an *input* string.
Application: String Matching

Find all the occurrences of a *target* string in an *input* string.

"the best of times"
Find all the occurrences of a target string in an input string.

Target “es” matches at [6, 16].
**Application:** String Matching

**Verification Time:**
- 0
- 200
- 400
- 600
- 800

**Exec Spec Proof**
- 669LoC
- 669LoC
- 285LoC
- 285LoC
- 180LoC
- 180LoC

**Application:** String Matching

**LoC (Proofs/Exec):** 5x

**Verification Time:** 20 min

**Human Effort:** 2 months
**LoC (Proofs/Exec):**

- Exec: 180LoC vs. 122LoC
- Spec: 285LoC vs. 248LoC
- Proof: 669LoC vs. 766LoC

**Verification Time:**

- Exec: 20 min
- Spec: 8x
- Proof: 38 sec

**Human Effort:**

- Exec: 2 months
- Spec: 8x
- Proof: 2 weeks
Haskell VS. Non-Haskell Proofs
Haskell VS. Non-Haskell Proofs

SMT- VS. Tactic- Based Automations
Haskell VS. Non-Haskell Proofs
SMT- VS. Tactic- Based Automations
Intrinsic VS. Extrinsic Verification
Intrinsic VS. Extrinsic Verification

\[
\text{take} :: \ i : \text{Nat} \rightarrow \ xs : \{i \leq \text{len} \ xs\} \rightarrow \{v | \text{len} \ v = i\}
\]

\[
\text{take } 0 \ _ = [\]
\]

\[
\text{take } i \ xs = x : \text{take} \ (i - 1) \ xs
\]
Intrinsic VS. Extrinsic Verification

\[
\text{take} :: i : \text{Nat} \to xs : \{i \leq \text{len } xs\} \to \{v \mid \text{len } v = i\}
\]

\[
\text{take } 0 \_ = []
\]

\[
\text{take } i \ \text{x}s = x : \text{take } (i-1) \ \text{x}s
\]

Definition \(\text{take} := \text{seq.takes.}\)

Theorem \(\text{take_spec}:\)
\[\forall i \ x, i \leq \text{length } x \Rightarrow \text{length } (\text{take } i \ x) = i.\]
Haskell VS. Non-Haskell Proofs

SMT- VS. Tactic- Based Automations

Intrinsic VS. Extrinsic Verification
Haskell VS. Non-Haskell Proofs

SMT- VS. Tactic- Based Automations

Intrinsic VS. Extrinsic Verification

Semantic VS. Syntactic Termination
Semantic VS. Syntactic Termination

\[
\text{chunk} :: \mathit{i} : \text{Pos} \rightarrow \mathit{xs} : [a] \rightarrow [[a]] / [\text{len } \mathit{xs}]
\]
**Semantic VS. Syntactic Termination**

\[
\text{chunk} :: i : \text{Pos} \rightarrow \text{x}s : [a] \rightarrow [[a]] \lor \text{len xs}
\]

\[
\text{Fixpoint chunk} \quad \{ M : \text{Type} \} \quad (\text{fuel} : \text{nat}) \quad (i : \text{nat}) \quad (x : M) : \text{option} \quad (\text{list} \ M)
\]
**Big VS. Tiny** Trusted Code Base

```
chunk :: i:Pos → xs:[a] → [[a]] / [len xs]
```
**Big VS. Tiny** Trusted Code Base

.hs → ghc → SMT → OK/Error
Haskell VS. Non-Haskell Proofs

SMT- VS. Tactic- Based Automations

Intrinsic VS. Extrinsic Verification

Semantic VS. Syntactic Termination

Big VS. Tiny Trusted Code Base
Haskell VS. Non-Haskell Proofs
SMT- VS. Tactic- Based Automations
Intrinsic VS. Extrinsic Verification
Semantic VS. Syntactic Termination
Big VS. Tiny Trusted Code Base
Proof Verifier VS. Assistant
A Tale of Two Provers

Conclusion

Liquid Haskell is a promising prover, but needs a lot of Coq-inspired future work.
A Tale of Two Provers

Conclusion

**Liquid Haskell** is a promising prover, but needs a lot of **Coq**-inspired future work.

Fast “tactics”

Liquid GUI Proof Assistant

Hackage Sharing Proofs

Thanks!