Liquidate your Assets

Reasoning about resource usage in Liquid Haskell

M. Handley, **N. Vazou**, and G. Hutton
University of Nottingham and **IMDEA**

```haskell
insert :: Ord a => x:a -> xs:[a] -> [a]


insert x []   = [x]
insert x (y:ys)
  | x <= y     = x:y:ys
  | otherwise = y:insert x ys
```

Refinement types for length preservation

```
insert :: Ord a => x:a -> xs:[a]
       -> {os:[a]|len os == 1 + len xs}

insert x []   = [
insert x (y:ys)
  | x <= y    = x
  | otherwise = y:insert x ys
```

Ranjit'r "shallow" specs
induction

What about resources?
(here number of comparisons)

Tracking resources

```
insert :: Ord a => x:a -> xs:[a]
       -> {os:[a]|len os == 1 + len xs}
        cost o <= length xs
insert x []  = [x]
insert x (y:ys)
  | x <= y     = x:y:ys
  | otherwise = y:insert x ys
```

Tracking resources using The Tick data type

```
 insert :: Ord a => x:a -> xs:[a]
-> {o:Tick {os:[a]|len os == 1 + len xs}
          tcost o <= length xs }
 insert x []   = [x]
insert x (y:ys)
  | x <= y     = x:y:ys
  | otherwise = y:insert x ys
```

The Tick data type

```
data Tick a = Tick { tcost :: Nat,
                     tval  :: a  }
```

The Tick data type

```
data Tick a = Tick { tcost :: Nat,
                     tval  :: a  }
```

The Applicative Instance

```
pure :: x:a -> {t:Tick a | tcost t == 0}
pure x = Tick 0 x

(<*>) :: f:(Tick (a -> b)) -> x:Tick a
      -> {t:Tick b | tcost t == tcost x + tcost f}
Tick i f <*> Tick j x = Tick (i+j) (f x)
```

Zero resources using The Tick data type

```
 insert :: Ord a => x:a -> xs:[a]
-> {o:Tick {os:[a]|len os == 1 + len xs}
          tcost o <= length xs }
 insert x []  = pure [x]
insert x (y:ys)
  | x <= y     = pure (x:y:ys)
  | otherwise = (pure (y:)) <*> insert x ys
```

The Tick data type

```
data Tick a = Tick { tcost :: Nat,
                     tval  :: a  }
```

Resource tracking

```
step :: x:a -> {t:Tick a | tcost t == 1}
step x = Tick 1 x

(</>) :: f:(Tick (a -> b)) -> x:Tick a
      -> {t:Tick b | tcost t == 1 + tcost x + tcost f}
Tick i f </> Tick j x = Tick (1+i+j) (f x)
```

Actual resources using the Tick data type

```
 insert :: Ord a => x:a -> xs:[a]
-> {o:Tick {os:[a]|len os == 1 + len xs}
          tcost o <= length xs }
 insert x []  = pure [x]
insert x (y:ys)
  | x <= y     = step (x:y:ys)
  | otherwise = (pure (y:)) </> insert x ys
```

Let’s define insertion sort!

Resource Tracking using Refinement Types

Tick monad lets you track resources in refinement types

Problem:

The bind operation breaks automatic verification

Solutions:

Ghost variables Extrinsic Proofs

Extrinsic proofs can encode arbitrary relational properties

Benchmarks

	Property	Lines of code Exec.	Spec.	Proof
Laziness [Danielsson 2008]				
Insertion sort	$\textsc{Cost}(\mathit{lisort}\ xs) \leqslant \lvert xs \rvert$	12	8	0
Implicit queues	$\textsc{Cost}(\mathit{lsnoc}\ q\ x) = 5, \textsc{Cost}(\mathit{view}\ q) = 1$	50	14	0
Relational [Aguirre et al. 2017; Çiçek et al. 2017; Radiček et al. 2017]				
2D count	$\textsc{Cost}(2\mathit{DCount}\ \mathit{find}_1) \leqslant \textsc{Cost}(2\mathit{DCount}\ \mathit{find}_2)$	16	3	24
Binary counters	$\textsc{Cost}(\mathit{decr}\ k\ \mathit{tt}) = \textsc{Cost}(\mathit{incr}\ k\ \mathit{ff})$	26	21	21
Boolean expressions	$\textsc{NoShort}(e) \Rightarrow \textsc{Cost}(\mathit{eval}_1\ e) = \textsc{Cost}(\mathit{eval}_2\ e)$	28	2	13
Constant-time comparison	$\textsc{Cost}(\mathit{compare}\ p\ u\_1) = \textsc{Cost}(\mathit{compare}\ p\ u\_2)$	3	8	3
Insertion sort	$\textsc{Sorted}(xs) \Rightarrow \textsc{Cost}(\mathit{isort}\ xs) \leqslant \textsc{Cost}(\mathit{isort}\ ys)$	16	17	44
Memory allocation of length	$\textsc{Cost}(\mathit{length}_2\ xs) - \textsc{Cost}(\mathit{length}_1\ xs) = \mathit{length}\ xs$	10	4	6
Relational insertion sort	$\textsc{Cost}(\mathit{isort}\ xs) - \textsc{Cost}(\mathit{isort}\ ys) = \mathit{unsortedDiff}\ xs\ ys$	16	11	69
Relational merge sort	$\textsc{Cost}(\mathit{msort}\ xs) - \textsc{Cost}(\mathit{msort}\ ys) \leqslant \lvert xs \rvert\ (1 + \log_2(\mathit{diff}\ xs\ ys))$	23	25	59
Square and multiply	$\textsc{Cost}(\mathit{sam}\ t\ x\ l_1) - \textsc{Cost}(\mathit{sam}\ t\ x\ l_2) \leqslant t * \mathit{diff}\ l_1\ l_2$	3	8	3
Datatypes [Vazou et al. 2018]				
Append's monoid laws	*see example 5 of section 2*	12	10	74
Appending	$\textsc{Cost}(xs\ \underline{+\!+}\ ys) = \lvert xs \rvert$	8	3	0
Flattening	$\textsc{Perfect}(t) \Rightarrow \textsc{Cost}(\mathit{flattenOpt}\ t) = 2^{\lvert t \rvert} - 1$	5	18	45
Optimised-by-construction reverse	$\mathit{reverse}\ xs >\!\!\sim\!\!> \mathit{fastReverse}\ xs$	18	37	140
Reversing (naive)	$\textsc{Cost}(\mathit{reverse}\ xs) = \frac{\lvert xs \rvert^2}{2} + \frac{\lvert xs \rvert + 1}{2}$	9	7	22
Reversing (optimised)	$\textsc{Cost}(\mathit{fastReverse}\ xs) = \lvert xs \rvert$	5	8	0
Sorting				
Data.List.sort	$\textsc{Cost}(\mathit{ssort}\ xs) \leqslant 4\ \lvert xs \rvert \log_2 \lvert xs \rvert + \lvert xs \rvert$	39	49	107
Insertion sort	$\textsc{Cost}(\mathit{isort}\ xs) \leqslant \lvert xs \rvert^2$	8	10	33
Merge sort	$\frac{\lvert xs \rvert}{2} \log_2 \lvert xs \rvert \leqslant \textsc{Cost}(\mathit{msort}\ xs) \leqslant \lvert xs \rvert \log_2 \frac{\lvert xs \rvert}{2} + \lvert xs \rvert$	22	69	139
Quicksort	$\textsc{Cost}(\mathit{qsort}\ xs) \leqslant \frac{1}{2}(\lvert xs \rvert + 1)(\lvert xs \rvert + 2)$	15	8	27
Total		344	340	829

Resource Tracking using Refinement Types

Tick monad lets you track resources in refinement types

Problem:

The bind operation breaks automatic verification

Solutions:

Ghost variables Extrinsic Proofs

Extrinsic proofs can encode arbitrary relational properties

Thanks!

The End

Metatheory

A corollary of monadic encapsulation + metatheory of refinement types:

THEOREM (SOUNDNESS OF COST ANALYSIS). *Let* $p :: Int \to Bool$ *be a predicate over integers and* $f :: x : \tau_x \to \tau$ *a safe and terminating function.*

- ***Intrinsic cost analysis*** *If* $\emptyset \vdash f :: x : \tau_x \to \{ t : Tick_\tau \mid p\ (tcost_\tau\ t) \}$, *then for all* $e_x \in [\![\tau_x]\!]$, $e_f\ e_x \hookrightarrow^\star Tick_\tau\ i\ e$ *and* $p\ i \hookrightarrow^\star true$.
- ***Extrinsic cost analysis*** *If* $\emptyset \vdash e :: x : \tau_x \to \{ v : \tau \mid p\ (tcost_\tau\ f\ x) \}$, *then for all* $e_x \in [\![\tau_x]\!]$, $f\ e_x \hookrightarrow^\star Tick_\tau\ i\ e$ *and* $p\ i \hookrightarrow^\star true$.

Future Directions

Resource Bound Inference

Automate lifting or, at least, erasure

Turn Tick into a monad transformer (e.g., to combine with Parallel Monad)