Algebraic Structures
CS 152 (Spring 2020)

Harvard University

Tuesday, April 7, 2020
Announcements

- HW2: Grades available in Gradescope
- HW3: grading in progress...
- HW4: due Apr 14
- HW5: will be released Apr 14, due May 1, and combine previous HW5 and HW6.
- Survey: by the end of Wednesday Apr 8
  - https://forms.gle/FM7mb9n4Gbze14Js6
Today, we will learn about

- Type constructors
  - Lists, Options

- Algebraic structures
  - Monoids
  - Functors
  - Monads

- Algebraic structures in Haskell
Type Constructors

▶ A type constructor creates new types from existing types

▶ E.g., product types, sum types, reference types, function types, ...
Assume CBV $\lambda$-calc with booleans, fixpoint operator $\mu x : \tau. \ e$

Expressions  

\[ \begin{align*} 
  e ::= & \cdots | \ [ \] \\
  & | e_1 :: e_2 | \text{isempty? } e | \text{head } e \\
  & | \text{tail } e 
\end{align*} \]

Values  

\[ \begin{align*} 
  v ::= & \cdots | \ [ \] | v_1 :: v_2 
\end{align*} \]

Types  

\[ \begin{align*} 
  \tau ::= & \cdots | \tau \text{ list} 
\end{align*} \]

Eval contexts  

\[ \begin{align*} 
  E ::= & \cdots | E :: e | v :: E \\
  & | \text{isempty? } E | \text{head } E | \text{tail } E 
\end{align*} \]
List inference rules

<table>
<thead>
<tr>
<th>Rule</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>isempty? <a href=""></a> → true</td>
<td>isempty? v₁ :: v₂ → false</td>
</tr>
<tr>
<td>head v₁ :: v₂ → v₁</td>
<td>tail v₁ :: v₂ → v₂</td>
</tr>
<tr>
<td>Γ ⊢ [] : τ list</td>
<td>Γ ⊢ e₁ : τ list</td>
</tr>
<tr>
<td>Γ ⊢ e : τ list</td>
<td>Γ ⊢ e : τ list</td>
</tr>
<tr>
<td>Γ ⊢ isempty? e : bool</td>
<td>Γ ⊢ head e : τ list</td>
</tr>
<tr>
<td>Γ ⊢ tail e : τ list</td>
<td>Γ ⊢ tail e : τ list</td>
</tr>
</tbody>
</table>

 append ≡ μf : τ list → τ list. λa : τ list. λb : τ list. if isempty? a then b else (head a) :: (f (tail a) b)
Options

Expressions  \( e ::= \cdots \mid \text{none} \mid \text{some } e \mid \text{case } e_1 \text{ of } e_2 \mid e_3 \)

Values  \( v ::= \cdots \mid \text{none} \mid \text{some } v \)

Types  \( \tau ::= \cdots \mid \tau \text{ option} \)

Eval contexts  \( E ::= \cdots \mid \text{some } E \mid \text{case } E \text{ of } e_2 \mid e_3 \)
Monoids
Monoid examples
Functors
Functor examples
Monad
Option monad
Algebraic structures in Haskell

- https://www.haskell.org/
- Pure functional language
- Call-by-need evaluation (aka lazy evaluation)
- Type classes: mechanism for ad hoc polymorphism
  - Declares common functions that all types within class have
  - We will use to express algebraic structures in Haskell
Monoid
Monad
Using Monads
Why Monads?

- Monads are *very* useful in Haskell
- Haskell is pure: no side effects
- But side effects useful!
- **Monadic types cleanly and clearly express side effects computation may have**
- Monads force computation into sequence
- Monads as type classes capture underlying structure of computation
  - Reusable readable code that works for any monad