On the Structure of Interpreters
Operational Semantics

• Operational semantics describe how a language operates.
• Given by a set of inference rules.
• Use a step relation \( \Downarrow \)

\[
\langle e, E, S \rangle \Downarrow \langle v, S' \rangle \quad l = E(x) \\
\langle (= x \; e), E, S \rangle \Downarrow \langle v, S' + \{l \mapsto v\} \rangle \quad \text{(Assgn)}
\]
1. The state of the semantics are those things that appear on both sides of the stepping relation
2. Inputs appear only on the left
3. Outputs appear on the right
State

- The state abstracts those things that change over time as the program executes
  - For example the heap
- The state might contain zero, one, or many parts
  - The heap, the stack, the current handlers etc.
Interpreters

- Interpreters have more detail than operational semantics.
- They are always recursive over the syntax of the language.
- Some things are only inputs, because they remain constant. E.g. the environment that maps names to locations, since the location never changes.
- The state appears as both an input and an output. The output captures the change over time.
Example 1

- The interpreter for the stack machine from HW1

- step:: Stack Int -> Instr -> Stack Int
Example 2

• The interpreter from HW3

\[ \text{interpE :: Env \ (Env\ Addr, [Vname], Exp)} \rightarrow \text{Env\ Addr} \rightarrow \text{State} \rightarrow \text{Exp} \rightarrow \text{IO(Value, State)} \]
Example 3

• The Exception machine from HW #4

interpE :: Env (Stack,[Vname],Exp)  
  -> Env Address  
  -> State  
  -> Exp  
  -> IO(Value,State)

data State  
  = State Stack Heap  
  | Exception State Fname [Value]

Here the State is much more complicated, it even comes in two modes: Normal and Exceptional. Normal states have two components, a Stack and a Heap.
Operations on States

- Operations on states propagate exceptional state.

\[
delta f g \ (\text{State } st \ hp) = \text{State } (f \ st) \ (g \ hp)
\]
\[
delta f g \ (\text{Exception } st \ fname \ vs) = \\
\quad \text{Exception } st \ fname \ vs
\]

\[
\text{alloc} :: \text{Value} \to \text{State} \to (\text{Address},\text{State})
\]
\[
\text{alloc } v \ \text{state} \mid \text{exceptional } \text{state} = \\
\quad (\text{error } "\text{Exception State in alloc"},\text{state})
\]
\[
\text{alloc } v \ \text{state} = (\text{HAddr } addr,\delta \text{Heap } f \ \text{state})
\]
\[
\quad \text{where } addr = \text{length } (\text{heap } \text{state})
\]
\[
\quad \quad f \ \text{heap} = \text{heap} +\{ (v) \} 
\]
Threading

run state (Add x y) =
  do { (v1, state1) <- interpE funs vars state x
        ; (v2, state2) <- interpE funs vars state1 y
        ; return (numeric "+") (+) v1 v2, state2) }
Special Casing the state

- The interpreter may do special things on certain kinds of state

```haskell
interpE funs vars state exp = traceG run state exp where
    run (state@(Exception s f vs)) exp
        = return(Bad,state)
    run state (Int n)
        = return(IntV n,state)
    run state (Char c)
        = return(CharV c,state)
...
Summary

• The shape and operations on the State of an interpreter can be used to encode many kinds of language features.
  – Assignment
  – Allocation
  – Exceptions
  – Continuations