Lecture 4: Abstract Data Types

Exercises

- Adapt the preceding specification of stacks (LIFO, Last-In First-Out) to describe queues instead (FIFO).

- Adapt the preceding specification of stacks to account for bounded stacks, of maximum size capacity.
  - Hint: put becomes a partial function.
Formal stack expressions

\[\text{value} = \text{item} \{ \text{remove} \{ \text{put} \{ \text{remove} \{ \text{put} \{ \text{remove} \{ \text{put} \{ \text{put} \{ \text{new}, x8 \}, x7 \}, x6 \}\}, \text{item} \{ \text{remove} \{ \text{put} \{ \text{put} \{ \text{new}, x5 \}, x4 \}\}\}, x2 \}\}, x1 \}\}\]
Expression reduction

\[
\text{value} = \text{item} (\text{remove} (\text{put} (\text{put} (\text{put} (\text{put} (\text{new}, x8), x7), x6) , \text{item} (\text{remove} (\text{put} (\text{put} (\text{new}, x5), x4) ) )\text{, x2}) )\text{, x1}) )
\]

Stack 1

value = item (remove (put (put (put (put (new, x8), x7), x6) , item (remove (put (put (new, x5), x4) ) )\text{, x2}) )\text{, x1}) )
Expression reduction

value = item {
  remove {
    put {
      remove {
        put {
          remove {
            put (put (put (new, x8), x7), x6)
          , item {
            remove {
              put (put (new, x5), x4)
            }
          }
        }
      }
    }
  }
}

, x1
, x2

Stack 1

Stack 2

Expression reduction

value = item {
  remove {
    put {
      remove {
        put {
          remove {
            put (put (put (new, x8), x7), x6)
          , item {
            remove {
              put (put (new, x5), x4)
            }
          }
        }
      }
    }
  }
}

, x1
, x2

Stack 1

Stack 2

Expression reduction

value = item {
  remove {
    put {
      remove {
        put {
          remove {
            put (put (put (new, x8), x7), x6)
          , item {
            remove {
              put (put (new, x5), x4)
            }
          }
        }
      }
    }
  }
}

, x1
, x2

Stack 1

Stack 2

Expression reduction

value = item {
  remove {
    put {
      remove {
        put {
          remove {
            put (put (put (new, x8), x7), x6)
          , item {
            remove {
              put (put (new, x5), x4)
            }
          }
        }
      }
    }
  }
}

, x1
, x2

Stack 1

Stack 2
Expression reduction

value = item
remove
put
remove
put
remove
put
item
remove
put
item
remove
put
item
remove
put
new, x8
, x7
, x6
, x5
, x4
, x2
, x1
Expression reduction

value = item
  remove {  
    put { 
      remove { 
        put ( 
          remove { 
            put (put (new, x8), x7), x6) 
          , item { 
            remove { 
              put (put (new, x5), x4) 
            } 
            , x2} 
          ) 
        , x1} 
      , x2} 
    ) 
  }

Stack 1
Expression reduction

```plaintext
value = item {
  remove {
    put {
      remove {
        put {
          remove {
            put (put (put (new, x8), x7), x6)
          , item {
            remove {
              put (put (new, x5), x4)
            }
          }
        , x2}
      }
    }
  }
  , x1}
}
```

Stack 1

```
x8
x7
x6
```
Expressed differently

\[
\begin{align*}
\text{value} &= \text{item}(\text{remove}(\text{put}(\text{remove}(\text{put}(\text{put}(\text{put}(\text{new}, x_8), x_7), x_8)), x_7), x_7)) \\
\text{s}_1 &= \text{new} \\
\text{s}_2 &= \text{put}(\text{put}(x_1, x_8), x_7) \\
\text{s}_3 &= \text{remove}(\text{put}(x_1, x_8)) \\
\text{s}_4 &= \text{item}(x_5) \\
\text{s}_5 &= \text{item}(x_5) \\
\text{s}_6 &= \text{item}(x_5) \\
\text{s}_7 &= \text{item}(x_5) \\
\text{s}_8 &= \text{item}(x_5) \\
\text{s}_9 &= \text{item}(x_5) \\
\text{s}_{10} &= \text{item}(x_5) \\
\text{s}_{11} &= \text{item}(x_5) \\
\text{value} &= \text{item}(\text{remove}(\text{put}(\text{put}(\text{put}(\text{put}(\text{put}(\text{put}(\text{new}, x_8), x_7), x_8)), x_7), x_7), x_7))
\end{align*}
\]

Sufficient completeness

- Three forms of functions in the specification of an ADT T:
  - Creators: \( \text{OTHER} \rightarrow T \) e.g. new
  - Queries: \( T \times \ldots \rightarrow \text{OTHER} \) e.g. item, empty
  - Commands: \( T \times \ldots \rightarrow T \) e.g. put, remove

- Sufficiently complete specification: a "Query Expression" of the form:
  \[ f (\ldots) \]
  where \( f \) is a query, may be reduced through application of the axioms to a form not involving \( T \).
Stack: An Abstract Data Type

- Types:
  \[ \text{STACK}[G] \]
  \( G \): Formal generic parameter

- Functions (Operations):
  - put: \( \text{STACK}[G] \times G \to \text{STACK}[G] \)
  - remove: \( \text{STACK}[G] \to \text{STACK}[G] \)
  - item: \( \text{STACK}[G] \to G \)
  - empty: \( \text{STACK}[G] \to \text{BOOLEAN} \)
  - new: \( \text{STACK}[G] \)

ADTs and software architecture

Abstract data types provide an ideal basis for modularizing software.
- Build each module as an implementation of an ADT:
  - Implements a set of objects with same interface
  - Interface is defined by a set of operations (the ADT's functions) constrained by abstract properties (its axioms and preconditions).
- The module consists of:
  - A representation for the ADT
  - An implementation for each of its operations
  - Possibly, auxiliary operations

Implementing an ADT

- Three components:
  - (E1) The ADT's specification: functions, axioms, preconditions.
    (Example: stacks.)
  - (E2) Some representation choice.
    (Example: \{representation, count\}.)
  - (E3) A set of subprograms (routines) and attributes, each implementing one of the functions of the ADT specification (E1) in terms of the chosen representation (E2).
    (Example: routines put, remove, item, empty, new.)
A choice of stack representation

“Push” operation:
\[
\begin{align*}
\text{count} & := \text{count} + 1 \\
\text{representation}[\text{count}] & := x
\end{align*}
\]

Application to information hiding

Public part:
- ADT specification (E1)

Secret part:
- Choice of representation (E2)
- Implementation of functions by features (E3)

Object technology: A first definition

- Object-oriented software construction is the approach to system structuring that bases the architecture of software systems on the types of objects they manipulate — not on "the" function they achieve.
Object technology: More precise definition

- Object-oriented software construction is the construction of software systems as structured collections of (possibly partial) abstract data type implementations.

Classes: The fundamental structure

- Merging of the notions of module and type:
  - Module = Unit of decomposition: set of services
  - Type = Description of a set of run-time objects ("instances" of the type)

- The connection:
  - The services offered by the class, viewed as a module, are the operations available on the instances of the class, viewed as a type.

Class relations

- Two relations:
  - Client
  - Heir
Overall system structure

A very deferred class

defered class
  COUNTER
  feature
    item : INTEGER is -- Counter value
      deferred
        up => -- Increase item by 1.
          deferred ensure
            item = old item + 1
          end
        down => -- Decrease item by 1.
          deferred ensure
            item = old item - 1
          end
    invariant
      item >= 0
  end

End of lecture 4