A Day of Revelations

- What's the difference between implementing an extension and a kernel?
- What are convention and correspondence? What are they for? Why do we bother?
- What about the rule that kernel operations cannot call other kernel operations?

(We were not drunk when we made it up. Seriously!)

Extension vs. Kernel

- Extension
  - Usually layered over a kernel
  - Does not access representation directly
- Kernel
  - Defines representation
  - Directly accesses representation
The Correspondence

- Consider the implementation of Sorting_Machine_Kernel with Heapsort
  - What's the math model of Sorting_Machine?
  - What's the representation?
  - What's the math model of the representation?

Math Model of Sorting_Machine

Sorting_Machine_Type is modeled by

```
( inserting: boolean
  contents: multiset of Item
)
```

self.inserting

self.contents
**Representation Math Model**

Rep is modeled by:

- `inserting_rep`: boolean
- `queue`: string of Item
- `array`: ARRAY_MODEL
- `heap_size`: integer

Math subtype `ARRAY_MODEL` is:

- `lb`: integer
- `ub`: integer
- `table`: INDEXED_TABLE

Math subtype `INDEXED_TABLE` is:

finite set of:

- `i`: integer
- `x`: Item

**Correspondence Continued...**

Sorting_Machine_Type is modeled by:

- `inserting`: boolean
- `contents`: multiset of Item

<table>
<thead>
<tr>
<th>abstract state space</th>
<th>concrete state space</th>
</tr>
</thead>
<tbody>
<tr>
<td>(true, {1, 2, 3})</td>
<td>?</td>
</tr>
<tr>
<td>(true, &lt;2, 1, 3&gt;, (1, 0, {}), 0)</td>
<td></td>
</tr>
</tbody>
</table>
Correspondence Continued...

- Given the Sorting_Machine m with value m = (true, {1, 2, 3}), what's the corresponding value for the representation? How do you know?
- Conversely, given the representation (true, <2, 1, 3>, (1, 0, {}), 0), what Sorting_Machine value does it represent? How do you know?

Correspondence Continued...

- The correspondence provides the mapping between the abstract value (the sorting machine) and the concrete value (the representation)
**Sorting Machine with Heapsort: Correspondence**

```python
self.inserting = self.inserting_rep and
if self.inserting
then self.contents = elements (self.queue)
else for all x: Item
    (x is in self.contents iff there exists i: integer
     (self.array.lb <= i and
      i <= self.heap_size and
      (i,x) is in self.array.table))
```

**The Convention**

- What Sorting Machine is represented by
  - (true, <2, 1, 3>, (1, 0, {}), 5)
  - (true, <2, 1, 3>, (1, 2, {(1, 7), (2, 8)}), 3)
  - (false, <>, (1, 2, {(1, 7), (2, 8)}), 3)?

- Is the following a correct implementation of Size? What would you need to know to answer this question?

```python
{ return self[heap_size]; }
```
The convention states any implementation-wide assumptions the implementer makes. It restricts the space of possible representation values. It allows us to reason about the correctness of each operation independently of the others.

Sorting_Machine_Type is modeled by:

- inserting: boolean
- contents: multiset of Item

Rep is modeled by:

- inserting_rep: boolean
- queue: string of Item
- array: ARRAY_MODEL
- heap_size: integer

Convention Continued...
**Sorting Machine with Heapsort: Convention**

if self.inserting_rep
then self.heap_size = 0 and
    self.array = (1, 0, empty_set)
else self.array.lb = 1 and
    0 <= self.heap_size and
    self.heap_size <= self.array.ub and
    SUB_TREE_IS_ORDERED (self.array, 1, self.heap_size) and
    self.queue = empty_string

**Convention: One More Thing**

- In implementing a kernel component, the implementer assumes that the convention holds at the beginning of each kernel operation
- It is the responsibility of the implementer to ensure that the convention holds at the end of each kernel operation
Convention Continued...

- What happens if in the middle of a kernel operation (where the convention may not hold) we call another kernel operation?