Verifying Imperative Abstractions with Dependent and Linear Types

Aaron Stump$^1$    Evan Austin$^2$

$^1$The University of Iowa

$^2$The University of Kansas

Funding from NSF CAREER.
The GURU Approach

<table>
<thead>
<tr>
<th>Industrial code</th>
<th>GURU</th>
<th>Math. functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>General recursion</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unaliased mutable state</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Aliased mutable state [new!]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>No concurrency</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Guru at a High-Level

- Pure functional language + logical theory.
- Terms : Types.
- Proofs : Formulas.
- Declare types, write code:
  
  \[
  (\text{append} \ [\ ] \ l') = l' \\
  (\text{append} \ x :: l \ l') = x :: (\text{append} \ l \ l')
  \]

- Prove theorems:
  
  \[
  \text{Forall}(A:\text{type})(l \ l' : <\text{list} \ A>). \\
  \{ (\text{length} \ (\text{append} \ l \ l')) = (\text{plus} \ (\text{length} \ l) \ (\text{length} \ l')) \}
  \]

- Define rich types:
  
  - \(<\text{vec} \ A \ N>\) – the type for vectors of \(A\)s of length \(N\).
  - So \(['a' 'b' 'c'] : <\text{vec} \ char \ 3>\).
I/O, mutable arrays, cyclic structures, etc.

Do not fit well into pure FP.

Approach: functional modeling.

- Define a pure functional model (e.g., vectors for arrays).
- Model is faithful, but slow.
- Use during reasoning.
- Replace with imperative code during compilation.
- Use *linear* (aka *unique*) types to keep in synch.
Example: Word-Indexed Mutable Arrays

- **Types:** `<\text{warray} A N L>`.  
  - \(A\) is type of elements.  
  - \(N\) is length of array.  
  - \(L\) is list of initialized locations.

- \((\text{new\_array} \ A \ N) : <\text{warray} A N []>\).

- **Writing to index** \(i\):  
  - requires proof: \(i < N\).  
  - functional model: consume old array, produce updated one.  
  - imperative implementation: just do the assignment.  
  - array’s type changes: `<\text{warray} A N i:::L>`.

- **Reading from index** \(i\):  
  - does not consume array.  
  - requires proof: \(i \in L\).
Example: FIFO Queues

- Mutable singly-linked list, with direct pointer to end.
- **Aliasing!**
- **GURU** approach: *heaplets*.
  - functionally model part of heap.
  - functional model: heaplet is list of aliased values.
  - implementation: no explicit heaplet.
  - functional model: aliases are indices into list.
  - implementation: aliases are reference-counted pointers.
  - caveat: not suitable for cyclic structures.
Run-times

- Linearity => memory deallocated explicitly.
- Typing ensures memory safety.
- **GURU**: no garbage collection!
- Leads to good performance (cf. [Xian, Srisa-an, Jiang 08]).

Benchmark: push all words in “War and Peace” through 2 queues.

<table>
<thead>
<tr>
<th>Language</th>
<th>Wallclock time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>HASKELL</strong> (DATA.QUEUE)</td>
<td>29.8</td>
</tr>
<tr>
<td><strong>HASKELL</strong> (DATA.SEQUENCE)</td>
<td>5.6</td>
</tr>
<tr>
<td>OCAML</td>
<td>1.3</td>
</tr>
<tr>
<td><strong>GURU</strong></td>
<td>1.0</td>
</tr>
</tbody>
</table>
Conclusion

- **GURU** combines FP, proofs, rich types.
- Linear types + dependent types ⇒ verified imperative abstractions.
- Mutable arrays, FIFO queues.
- More examples to come.
- Version 1.0 is close to release:

www.guru-lang.org