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

Industrial code	↞— GURU	Math. functions

General recursion
Unaliased mutable state
Aliased mutable state [new!]
No concurrency

GURU at a High-Level

- Pure functional language + logical theory.
- Terms : Types.
- Proofs : Formulas.
- Declare types, write code:

```
(append [] l') = l'
(append x::l l') = x::(append l l')
```

- Prove theorems:

```
Forall(A:type)(l l':<list A>).
 {(length (append l l')) = (plus (length l) (length l'))}
```

- Define rich types:
  - `<vec A N>` – the type for vectors of `A`s of length `N`.
  - So `['a' 'b' 'c'] : <vec char 3>`.

Functional Modeling for Imperative Abstractions

- 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: `<warray A N L>`.
  - `A` is type of elements.
  - `N` is length of array.
  - `L` is list of initialized locations.
- `(new_array A N) : <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: `<warray A N i::L>`.
- Reading from index `i`:
  - does not consume array.
  - requires proof: `i` ∈ `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.

Language	**Wallclock time (s)**
HASKELL (DATA.QUEUE)	29.8
HASKELL (DATA.SEQUENCE)	5.6
OCAML	1.3
GURU	1.0

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`