Refining Reasoning About Arrays in Formal Verification

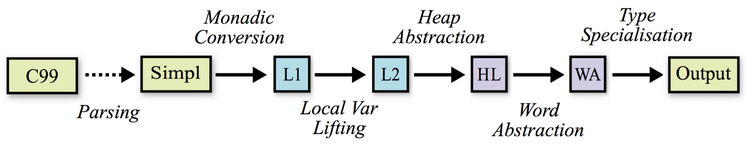
Nyle Rodgers, Yakoub Nemouchi, Peter Lammich, Binoy Ravindran
{nyle,nemouchi,lpeter1,binoy}@vt.edu

Overview

We developed a library allowing a refinement from C arrays to lists. The development effort is on a top of the interactive theorem prover Isabelle/HOL[1] and AutoCorres[2,3,4]. Refining reasoning about arrays to the level of lists was found to result in more modular and optimized proofs.

AutoCorres Overview

AutoCorres first translates C code into Simpl, a formal imperative language embedded in Isabelle/HOL. It then transforms Simpl into a monadic functional form to ease reasoning. Along the way, it generates correctness proofs for each refinement transformation.



The processing chain of AutoCorres. Figure from [3].

AutoCorres 32-Bit Heap Memory Model

Function	Description
`is_valid_w32 s p`	Check if the given pointer `p` refers to a valid 32-bit integer in memory state `s`
`heap_w32 s p`	Read the 32-bit word referenced by `p` in memory state `s`
`heap_w32_update f s`	Use the function `f` to produce a new memory state from state `s`

These functions satisfy the following properties

```
heap_w32_update f1 (heap_w32_update f2 s) =
  heap_w32_update (f1 o f2) s

heap_w32 (heap_w32_update f s) = f (heap_w32 s)

heap_w32_update id s = s

is_valid_w32 (heap_w32_update f s) p =
  is_valid_w32 s p
```

Array definition

The definition used to link arrays in memory to lists is

```
array s p data =
  (∀i < (length data).
     not_at_end (p +p i) ∧
     is_valid_w32 s (p +p i) ∧
     heap_w32 s (p +p i) = data ! i)
```

In English, this says that there is an array in the memory state `s`, starting at location `p`,corresponding to the list `data` iff for all `i` that are indices of `data`, the memory location at that index is not at the end of memory, holds a valid 32-bit word, and has the same value as the corresponding entry in `data`. The requirement that each address not be at the end of memory ensures that the array does not “wrap around” from the end of memory back to the start.

Read Array Definition

Additionally, the function `read_array` is defined as

```
read_array s p n =
  map (heap_w32 s) (map (λi. (p +p i)) [0 ..< n])
```

That is, it reads out the list corresponding to an array given the corresponding memory state, start address and length.

Rewriting Rules

Based on these definitions, if

```
array s p data ∧
i < length data ∧ j < length data ∧
n = length data
```

then the following rewriting rules can be applied

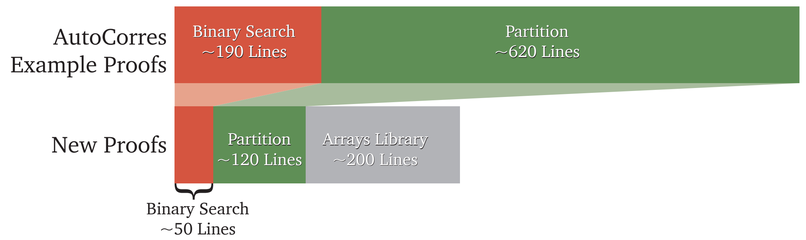
```
read_array s p n = data

length (read_array s p n) = n

heap_w32 s (p +p i) = data ! i

(p +p i = p +p j) = (i = j)
(p +p i ≠ p +p j) = (i ≠ j)
(p +p i < p +p j) = (i < j)
(p +p i ≤ p +p j) = (i ≤ j)

(read_array
  (heap_w32_update ((λa. a(p +p i := x))) s) p n)
  = read_array s p n)[i := x]
```

Case Study: Binary Search

The binary search function[5] uses a binary search algorithm to determine if a number is present in a sorted array. With the approach developed, the proof of correctness for the function was reduced from about 190 lines in the provided AutoCorres example to about 50 lines outside of the library.

Case Study: Partition

The partition function[5] re-arranges an array such that, for some pivot element, every element before the pivot in the array is less than the pivot, and every element after the pivot is greater than or equal to the pivot. This partition function is used in the quicksort implementation which is in the AutoCorres examples[5]; extending this approach to handle the additional requirements of the quicksort function is still a work in progress. With the approach developed, the proof of correctness for the function was reduced from about 620 lines in the example to about 120 lines outside of the library.



Visualization of the approach’s benifits for proof length

References

[1] T. Nipkow, L. Paulson, and M. Wenzel. Isabelle/HOL — A Proof Assistant for Higher-Order Logic, volume 2283 of LNCS. Springer, 2002.
[2] David Greenaway, June Andronick, and Gerwin Klein. Bridging the gap: Automatic verified abstraction of C. Proceedings of the Third International Conference on Interactive Theorem Proving (ITP), August 2012. https://ts.data61.csiro.au/publications/nicta_full_text/5662.pdf
[3] David Greenaway, Japheth Lim, June Andronick, and Gerwin Klein. Don’t sweat the small stuff—Formal verification of C code without the pain. Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation. ACM, June 2014. https://ts.data61.csiro.au/publications/nicta_full_text/7629.pdf
[4] https://ts.data61.csiro.au/projects/TS/autocorres/
[5] https://github.com/seL4/l4v/blob/master/tools/autocorres/tests/examples/