Reverse Abstraction in ACL2

Dr. Bill Young
Computer Sciences Department
University of Texas at Austin
byoung@cs.utexas.edu

ACL2 Workshop 2004

Draft of November 8, 2004

Formal Modeling

Formal models of digital systems are constructed for a variety of purposes.

Simulator models: may be highly optimized for efficiency, but not congenial for proof;

Abstract models: may be elegant and well-suited for formal analysis, but highly inefficient for execution.

It may be difficult to build a single model that supports such disparate goals.

Possible Solutions

- Construct an abstract system model, and then refine it through a series of steps to eke out execution efficiency.
- Introduce conceptual abstractions into an existing low-level model hand-tooled for efficiency.

AA 7 odel

The existing artifact for this project is the Rockwell Collins AAMP7 processor model.

- Very detailed low level model of the AAMP7 processor.
- Represents man-years of effort.
- Highly optimized for efficient execution.
- Extensive use of sophisticated macros.

7 Operation Semantics

Operation semantics are define in terms of a complex *reader* macro, that essentially emulates an imperative language in an applicative context.

Example: the `LIT16` operation takes a 16-bit quantity from the instruction stream and pushes it onto the stack.

```
(defun op-lit16 (st)
  (declare (xargs :stobjs (st)))
  (AAMP *state->state*
        (fetch_word ux);
        (push ux);
        st))
```

peration Semantics

The call `(OP-LIT16 ST)` actually macro-expands into the following:

```
(update-nth
   *aamp.ram*
   (write_memory
      (makeaddr (nth *aamp.denvr* st)
                (logand 65535
                        (logext 32 (+ -2 (nth *aamp.tos* st)))))
      (gacc::rx 16
                (makeaddr (nth *aamp.cenvr* st)
                          (nth *aamp.pc* st))
                (nth *aamp.ram* st))
      (nth *aamp.ram* st))
   (update-nth
      *aamp.tos*
      (logand 65535
              (logext 32 (+ -2 (nth *aamp.tos* st))))
      (update-nth *aamp.pc*
                  (logand 65535
                          (logext 32 (+ 2 (nth *aamp.pc* st))))
                  st)))
```

bstracting

Staring at the specification we notice the form:

```
(logand 65535 (logext 32 (+ k x)))
```

This is provably equivalent to the slightly simpler logical expression:

```
(loghead 16 (+ k x)).
```

and we could rewrite it to this form, but that still isn't very abstract.

bstracting

Let's define the following function and rewrite rule:

```
(defun plus16 (k x)
  (loghead 16 (+ k x)))

(defthm plus16-abstractor
  (equal (loghead 16 (+ k x))
         (plus16 k x)))
```

Note that it would be disastrous to have both of these enabled.

Defabstractor

This process is very stylized and can all be accomplished with a macro.

```
(defabstractor plus16 (k x)
  (loghead 16 (+ k x)))
```

which encapsulates the definition of `PLUS16`, rewrite rule, and disable.

ultiple Forms

If there are various forms of the same essential abstract concept, we can "canonicalize" them:

```
(defthm plus16-abstractor-2
  (equal (logand 65535 (add32 x k))
         (plus16 k x)))
```

Abstractions may be nested.

```
(defabstractor next-stack-address (st)
  (makeaddr (nth *aamp.denvr* st)
            (plus16 -2 (nth *aamp.tos* st))))
```

Rewriting with bstractions

Once the abstractions are in place, other rewrites are suggested, e.g., to consolidate multiple updates to the state:

```
(defthm inc-pc-inc-pc
  (implies (and (st-p st)
                (unsigned-byte-p 16 (+ i j (pc st))))
           (equal (inc-pc i (inc-pc j st))
                  (inc-pc (+ i j) st))))
```

pplying everse bstraction

Applying reverse abstraction and rewriting to the OP-LIT16 semantics, we can prove:

```
(defthm lit16-rewriter
  (implies
     (st-p st)
     (equal (op-lit16 st)
            (write-to-ram (next-stack-address st)
                          (fetch-code-word (pc st)
                                           (cenvr st)
                                           (ram st))
                          (inc-tos -2 (inc-pc 2 st))))))
```

This provides an alternative semantics for the `LIT16` operation.

Efficiency

Emulation of iterative behavior in an applicative context may be very inefficient. Think about the computation of the top-of-stack pointer in:

```
(defun op-addi (st)
  (reader
    '( (fetch-word x)
       (push x)
       (fetch-word y)
       (push y)
       (add)
     )))
```

Naively, you increment twice and then decrement. An abstract implementation merely increments once.

Conclusions

The ultimate goal is to be able to prove properties of AAMP7 programs. The reverse abstraction process is a useful step toward a suitable semantics.

- We have described an approach to introduce "abstraction" into an existing formal specification.
- The result may actually be more efficient to execute because optimizations are easier to see in the abstract version.
- The result is more readable and hopefully more amenable to formal analysis.