Proving Unbounded Theorems with the Help of GL

Cuong Chau

Matt Kaufmann

Agenda

- A technique for proving unbounded theorems with the help of GL.

- Benefit of using that technique in certifying the 32-bit physical memory model.

Agenda

- A technique for proving unbounded theorems with the help of GL.

- Benefit of using that technique in certifying the 32-bit physical memory model.

Approach

- GL is a symbolic simulation framework for proving bounded ACL2 theorems. It cannot prove theorems that contain unbounded variables.

Approach

- GL is a symbolic simulation framework for proving bounded ACL2 theorems. It cannot prove theorems that contain unbounded variables.
- Proving unbounded theorems might not trivial if they are complicated. However, if they can be transformed into bounded theorems, then we can use GL to solve the problem.

Approach

- GL is a symbolic simulation framework for proving bounded ACL2 theorems. It cannot prove theorems that contain unbounded variables.
- Proving unbounded theorems might not trivial if they are complicated. However, if they can be transformed into bounded theorems, then we can use GL to solve the problem.
- Present a trick that proves unbounded theorems with the help of GL.

Simple Example

- (implies (integerp x)
  (equal (mod x 8)
  (logand x 7))))

Although x is unbounded, only its 3 low bits affect the computation in the above theorem. So, we can transform it to the bounded lemma:

- (implies (integerp x)
  (equal (mod x[3:0] 8)
  (logand x[3:0] 7))))

where x[j:i] represents the bit string from index i to j of x (0 <= i <= j).

Simple Example

- (implies (integerp x)
  (equal (mod x 8)
  (logand x 7))))

Although x is unbounded, only its 3 low bits affect the computation in the above theorem. So, we can transform it to the bounded lemma:

- (implies (integerp x)
  (equal (mod x[3:0] 8)
  (logand x[3:0] 7))))

Then, the unbounded theorem will follow by applying two following rewrite rules:

- (equal (mod x[3:0] 8) (mod x 8))
- (equal (logand x[3:0] 7) (logand x 7))

Main Theorem

```
(defthm main
  (implies (and (natp i02)
                (<= i02 2))
           (equal (logior (mod (ash mem-val (* -8 i02))
                               *2^8*)
                          (* *2^8*
                             (mod (ash mem-val (+ -8 (* -8 i02)))
                                  *2^8*)))
                  (mod (ash mem-val (* -8 i02))
                       *2^16*))))
```

Main Theorem

```
(defthm main
  (implies (and (natp i02)
                (<= i02 2))
           (equal (logior E0

                    (* *2^8*
                       (mod (ash mem-val (+ -8 (* -8 i02)))
                            *2^8*)))
                  (mod (ash mem-val (* -8 i02))
                       *2^16*))))
```

Main Theorem

```
(defthm main
  (implies (and (natp i02)
                (<= i02 2))
           (equal (logior E0

                          (* *2^8*
                             E1))

                  (mod (ash mem-val (* -8 i02))
                       *2^16*))))
```

Main Theorem

```
(defthm main
  (implies (and (natp i02)
                (<= i02 2))
           (equal (logior E0

                          (* *2^8*
                             E1))

                  E2)))
```

Analyze E0

- E0 = (mod (ash mem-val (* -8 i02))
  2^8)
  = mem-val[(+ 7 (* 8 i02)) : (* 8 i02)]

- (and (<= 0 i02) (<= i02 2)) => (and (<= 0 (* 8 i02)) (<= (* 8 i02) 16) (<= 7 (+ 7 (* 8 i02))) (<= (+ 7 (* 8 i02)) 23))

Analyze E0

- E0 = (mod (ash mem-val (* -8 i02))
  2^8)
  = mem-val[(+ 7 (* 8 i02)) : (* 8 i02)]

- (and (<= 0 i02)
  (<= i02 2))
  =>
  (and (<= 0 (* 8 i02))
  (<= (* 8 i02) 16)
  (<= 7 (+ 7 (* 8 i02)))
  (<= (+ 7 (* 8 i02)) 23))

=> Only mem-val[23 : 0] of mem-val affects E0.

Analyze E1

- E1 = (mod (ash mem-val (+ -8 (* -8 i02)))
  2^8)
  = mem-val[(+ 15 (* 8 i02)) : (+ 8 (* 8 i02))]

Analyze E1

- E1 = (mod (ash mem-val (+ -8 (* -8 i02)))
  2^8)
  = mem-val[(+ 15 (* 8 i02)) : (+ 8 (* 8 i02))]

- (and (<= 0 i02) => (and (<= 8 (+ 8 (* 8 i02)))
  (<= i02 2)) (<= (+ 8 (* 8 i02)) 24)
  (<= 15 (+ 15 (* 8 i02)))
  (<= (+ 15 (* 8 i02)) 31))

=> Only mem-val[31 : 8] of mem-val affects E1.

Analyze E2

- E2 = (mod (ash mem-val (* -8 i02))
  2^16)
  = mem-val[(+ 15 (* 8 i02)) : (* 8 i02)]

Analyze E2

- E2 = (mod (ash mem-val (* -8 i02))
  2^16)
  = mem-val[(+ 15 (* 8 i02)) : (* 8 i02)]

- (and (<= 0 i02)   =>   (and (<= 0 (* 8 i02))
  (<= i02 2))   (<= (* 8 i02) 16)
  (<= 15 (+ 15 (* 8 i02)))
  (<= (+ 15 (* 8 i02)) 31))

=> Only mem-val[31 : 0] of mem-val affects E2.

Claim

- Only mem-val[23 : 0] of mem-val affects E0.
- Only mem-val[31 : 8] of mem-val affects E1.
- Only mem-val[31 : 0] of mem-val affects E2.

$\Rightarrow$ Only mem-val[31 : 0] of mem-val affects E0, E1, and E2.

Claim

- Only mem-val[23 : 0] of mem-val affects E0.
- Only mem-val[31 : 8] of mem-val affects E1.
- Only mem-val[31 : 0] of mem-val affects E2.

$\Rightarrow$ Only mem-val[31 : 0] of mem-val affects E0, E1, and E2.

- mem-val[31 : 0]

 = (mod mem-val *2^32*)

 = (logand mem-val *2^32-1*)

 = …

Claim

- Only mem-val[23 : 0] of mem-val affects E0.
- Only mem-val[31 : 8] of mem-val affects E1.
- Only mem-val[31 : 0] of mem-val affects E2.

$\Rightarrow$ Only mem-val[31 : 0] of mem-val affects E0, E1, and E2.

$\Rightarrow$ The main theorem can be transformed into the bounded lemma by replacing mem-val by mem-val[31 : 0] in the main theorem.

Bounded Main-2 Lemma

```
(defthm main-2
  (let ((mem-val (mod mem-val *2^32*)))
    (implies (and (natp i02)
                  (< i02 3))
             (equal (logior (mod (ash mem-val (* -8 i02))
                                 *2^8*)
                            (* *2^8*
                               (mod (ash mem-val (+ -8 (* -8 i02)))
                                    *2^8*)))
                    (mod (ash mem-val (* -8 i02))
                         *2^16*)))))
```

Bounded Main-1 Lemma

```
(def-gl-thm main-1
  :hyp (and (natp i02)
              (< i02 3)
              (n32p mem-val))
  :concl (equal (logior (mod (ash mem-val (* -8 i02))
                                  *2^8*)
                              (* *2^8*
                                 (mod (ash mem-val (+ -8 (* -8 i02)))
                                      *2^8*)))
                      (mod (ash mem-val (* -8 i02))
                           *2^16*))
  :g-bindings
  `((mem-val (:g-number ,(gl-int 0 2 33)))
    (i02 (:g-number ,(gl-int 1 2 3)))))
```

Bounded Main-2 Lemma

```
(defthm main-2
  (let ((mem-val (mod mem-val *2^32*)))
    (implies (and (natp i02)
                  (< i02 3))
             (equal (logior (mod (ash mem-val (* -8 i02))
                                 *2^8*)
                            (* *2^8*
                               (mod (ash mem-val (+ -8 (* -8 i02)))
                                    *2^8*)))
                    (mod (ash mem-val (* -8 i02))
                         *2^16*)))))
```

Rewrite Rules

- (mod (ash (mod mem-val *2^32*) (* -8 i02))
  2^8)

= (mod (ash mem-val (* -8 i02))
  2^8)

= E0

Rewrite Rules

- (mod (ash (mod mem-val *2^32*) (* -8 i02))
  2^8)

= (mod (ash mem-val (* -8 i02))
  2^8)

= E0

- (mod (ash (mod mem-val *2^32*) (+ -8 (* -8 i02)))
  2^8)

= E1

Rewrite Rules

- (mod (ash (mod mem-val *2^32*) (* -8 i02))
  2^8)

= (mod (ash mem-val (* -8 i02))
  2^8)

= E0

- (mod (ash (mod mem-val *2^32*) (+ -8 (* -8 i02)))
  2^8)

= E1

- (mod (ash (mod mem-val *2^32*) (* -8 i02))
  2^16)

= E2

Main Theorem

```
(defthm main
  (let ((mem-val (mod mem-val *2^32*)))
   (implies (and (natp i02)
                 (< i02 3))
            (equal (logior (mod (ash mem-val (* -8 i02))
                                *2^8*)
                           (* *2^8*
                              (mod (ash mem-val (+ -8 (* -8 i02)))
                                   *2^8*)))
                   (mod (ash mem-val (* -8 i02))
                        *2^16*)))
   :hints (("Goal" :use (main-2))))
```

Agenda

- A technique for proving unbounded theorems with the help of GL.

- Benefit of using that technique in certifying the 32-bit physical memory model.

Benefit

- The main theorem will help to prove 16-bit read-over-write theorems in the 32-bit physical memory model without requiring the (x86p x86) hypothesis.

```
(defthm |rm-low-16 over wm-low-16 at diff-addrs & non-overlap|
  (implies (and (or (< (1+ addr1) addr2)
                    (< (1+ addr2) addr1))
                (n16p val)
                ~~(x86p x86)~~
                …)
           (equal (rm-low-16 addr2 (wm-low-16 addr1 val x86))
                  (rm-low-16 addr2 x86))))
```

16-Bit Read-Over-Write

```
(defthm |rm-low-16 over wm-low-16 at diff-addrs & non-overlap|
  (implies (and (or (< (1+ addr1) addr2)
                    (< (1+ addr2) addr1))
                (n16p val)
                (x86p x86)
                …)
           (equal (rm-low-16 addr2 (wm-low-16 addr1 val x86))
                  (rm-low-16 addr2 x86))))
```

- (rm-low-<i> addr2 x86) performs reading an <i>-bit value from addr2 in x86 memory field.
- (wm-low-<j> addr1 val x86) performs writing a <j>-bit value val into x86 memory field at addr1.

Supporting Lemma

```
(defthm rm-low-16-as-rm-low-08
  (implies (and (natp addr)
                (< (+ 1 addr) *mem-size-in-bytes*))
           (equal (rm-low-16 addr x86)
                  (let* ((byte0 (rm-low-08 addr x86))
                         (byte1 (rm-low-08 (+ 1 addr) x86)))
                    (logior (ash byte1 8)
                            byte0)))))
```

Key Checkpoint

```
(implies (and (natp addr)
              (< addr 4503599627370495)
              (<= (mod addr 4) 2)
              (integerp (memi (ash addr -2) x86)))
         (equal (mod (ash (memi (ash addr -2) x86)
                          (* -8 (mod addr 4)))
                     65536)
                (logior (mod (ash (memi (ash addr -2) x86)
                                  (* -8 (mod addr 4)))
                             256)
                        (* 256
                           (mod (ash (memi (ash addr -2) x86)
                                     (+ -8 (* -8 (mod addr 4))))
                                256)))))
```

Problem

- The key checkpoint is the main theorem we discussed earlier, where i02 is replaced with (mod addr 4), and mem-val is replaced with (memi (ash addr -2) x86).
- Although (memi (ash addr -2) x86) returns a 32-bit value, proving (n32p (memi (ash addr -2) x86)) requires (x86p x86) hypothesis by the following lemma:

```
(defthm memi-is-n32p
  (implies (and (x86p x86)
                (natp i)
                (< i *mem-size-in-dwords*))
           (n32p (memi i x86))))
```

Problem with (x86p x86)

- The present of (x86p x86) hypothesis in read-over-write and write-over-write theorems causes significant slowdown when proving lemmas containing read-over-long-nested-writes as well as write-over-long-nested-writes into memory.

=> The main theorem is a solution for avoiding (x86p x86) hypothesis in read-over-write theorems.

Problem with (x86p x86)

- The present of (x86p x86) hypothesis in read-over-write and write-over-write theorems causes significant slowdown when proving lemmas containing read-over-long-nested-writes as well as write-over-long-nested-writes into memory.

=> The main theorem is a solution for avoiding (x86p x86) hypothesis in read-over-write theorems.

- How about write-over-write theorems?

Supporting Lemma

```
(defthm wm-low-16-as-wm-low-08-lemma-1
  (implies (and (n02p i02) (< i02 3) (n16p val) ~~(n32p mem-val)~~)
           (equal (logior (* (mod (ash val -8) 256)
                              (expt 2 (+ 8 (* 8 i02))))
                          (logand (lognot (* 255 (expt 2 (+ 8 (* 8 i02)))))
                                  (* (mod val 256) (expt 256 i02)))
                          (logand (lognot (* 255 (expt 256 i02)))
                                  (lognot (* 255 (expt 2 (+ 8 (* 8 i02)))))
                                  mem-val))
                  (logior (* val (expt 256 i02))
                          (logand (lognot (* 65535 (expt 256 i02)))
                                  mem-val))))
```

Problem

- We cannot transform wm-low-16-as-wm-low-08-lemma-1 into a bounded lemma because the following condition is not satisfied:
  - Only fixed finite bits of unbounded variables affect the computation.

Supporting Lemma

(defthm wm-low-16-as-wm-low-08-lemma-1
(implies (and (n02p i02) (< i02 3) (n16p val) ~~(n32p mem-val)~~)
(equal (logior (* (mod (ash val -8) 256)
(expt 2 (+ 8 (* 8 i02))))
(logand (lognot (* 255 (expt 2 (+ 8 (* 8 i02)))))
(* (mod val 256) (expt 256 i02)))
(logand (lognot (* 255 (expt 256 i02)))
(lognot (* 255 (expt 2 (+ 8 (* 8 i02)))))
mem-val))
(logior (* val (expt 256 i02))
(logand (lognot (* 65535 (expt 256 i02)))
mem-val))))

Strategy

```
(defthm wm-low-16-as-wm-low-08-lemma-1
  (implies (and (n02p i02) (< i02 3) (n16p val) (n32p mem-val))
           (equal (logior (* (mod (ash val -8) 256)
                             (expt 2 (+ 8 (* 8 i02))))
                          (logand (lognot (* 255 (expt 2 (+ 8 (* 8 i02)))))
                                  (* (mod val 256) (expt 256 i02)))
                          (logand (lognot (* 255 (expt 256 i02)))
                                  (lognot (* 255 (expt 2 (+ 8 (* 8 i02)))))
                                  mem-val))
                  (logior (* val (expt 256 i02))
                          (logand (lognot (* 65535 (expt 256 i02)))
                                  mem-val))))
```

Strategy

```
(defthm wm-low-16-as-wm-low-08-lemma-1
  (implies (and (n02p i02) (< i02 3) (n16p val) ~~(n32p mem-val)~~)
           (equal (logior (* (mod (ash val -8) 256)
                             (expt 2 (+ 8 (* 8 i02))))
                          (logand (lognot (* 255 (expt 2 (+ 8 (* 8 i02)))))
                                  (* (mod val 256) (expt 256 i02))))
                          (logand (lognot (* 255 (expt 256 i02)))
                                  (lognot (* 255 (expt 2 (+ 8 (* 8 i02)))))
                                  mem-val))
                  (logior (* val (expt 256 i02))
                          (logand (lognot (* 65535 (expt 256 i02)))
                                  mem-val))))
```

Timing Results

The experiments below were performed on “eld” using /projects/acl2/svn-recent/ccl-saved_acl2hp

Certification time	**8-bit**	**32-bit**	**32-bit (x86p x86)**
Lemma loop-effects	29.90s	32.83s	498.17s
Lemma prime-effects	20.80s	22.84s	475.02s
Whole model	32:32.29s	34:35.85s	43:42.12s

Lemma loop-effects and prime-effects contain 8-bit read-over-80-nested-writes.

Questions!