Abstract Stobjs and Their Application to ISA Modeling

Shilpi Goel
Warren A. Hunt, Jr.
Matt Kaufmann

The University of Texas at Austin

30^{th} May, 2013

OUTLINE

OUTLINE

EXECUTION AND REASONING

- ACL2 development is geared towards:
  - Efficient execution
  - Effective reasoning

Execution and Reasoning

- ACL2 development is geared towards:
  - Efficient execution
  - Effective reasoning
- Abstract stobjs, introduced in ACL2 Version 5.0, are also in this tradition.

RUNNING EXAMPLE: Y86

- ISA Model: **y86**
  - 32-bit architecture
  - Academic simplification of the x86
  - Supporting Materials: `books/models/y86/y86-two-level-abs`

Running Example: Y86

- ISA Model: **y86**
  - 32-bit architecture
  - Academic simplification of the x86
  - Supporting Materials:
    `books/models/y86/y86-two-level-abs`
- We benefit from abstract stobjs in our x86 model too.

GOAL OF THIS TALK

- Will *not* be talking about the logical foundations of abstract stobjs today...
  - (For that, see the Essay on the Correctness of Abstract Stobjs in ACL2 source file `other-events.lisp`.)

GOAL OF THIS TALK

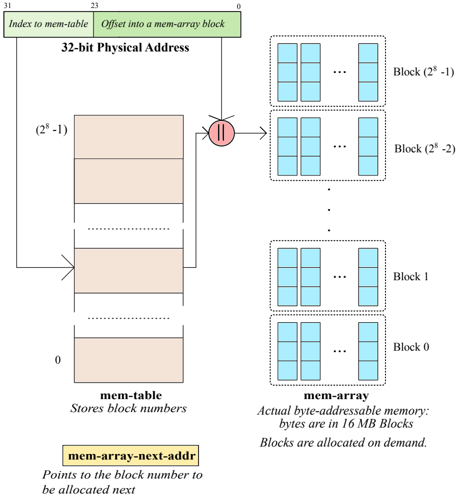
- Will *not* be talking about the logical foundations of abstract stobjs today...
  - (For that, see the Essay on the Correctness of Abstract Stobjs in ACL2 source file `other-events.lisp`.)
- Will introduce abstract stobjs to the ACL2 community so that users can consider using this feature in their proof developments

Y86 MODEL IN ACL2

- State: represented by a concrete stobj called `x86-32$c`

Component	Description
general-purpose registers	array of length 8; each element is a 32-bit unsigned number
instruction pointer	32-bit unsigned number
flags register	32-bit unsigned number
physical memory	a **space-efficient implementation** of 4 GB physical memory

Y86 MEMORY MODEL



[FMCAD'12] Hunt & Kaufmann: A Formal Model of a Large Memory that Supports Efficient Execution

Y86 STATE RECOGNIZER

- Stobj recognizer: a simple structural check for the stobj fields

Y86 STATE RECOGNIZER

- Stobj recognizer: a simple structural check for the stobj fields
- **What does it mean to have a good y86 state?**

Y86 STATE RECOGNIZER

- Stobj recognizer: a simple structural check for the stobj fields
- **What does it mean to have a good y86 state?**
  Stobj recognizer
  +
  an invariant stating that the space-efficient implementation of the memory gives a well-formed y86 memory

Y86 STATE RECOGNIZER

- Stobj recognizer: a simple structural check for the stobj fields
- **What does it mean to have a good y86 state?**
  Stobj recognizer
  +
  an invariant stating that the space-efficient implementation of the memory gives a well-formed y86 memory
- Have to carry around the complexities of the space-efficient memory model during proofs...

ABSTRACT STOBJS

An *abstract stobj* provides a simple logical interface to a *corresponding* concrete stobj.

ABSTRACT STOBJS

An *abstract stobj* provides a simple logical interface to a *corresponding* concrete stobj.

- Fast execution: provided by the previously-defined concrete stobj

Abstract Stobjs

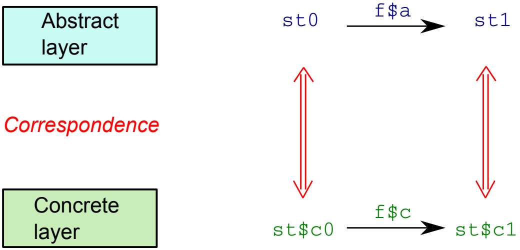
An *abstract stobj* provides a simple logical interface to a *corresponding* concrete stobj.

- Fast execution: provided by the previously-defined concrete stobj
- Effective reasoning: provided by an alternate (logical) representation of the concrete stobj

OUTLINE

OUTLINE

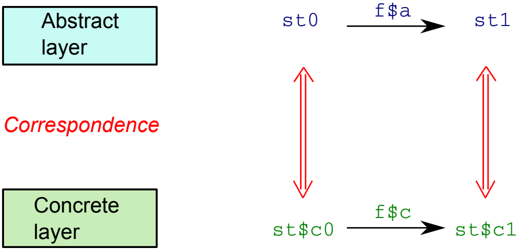
PROOF OBLIGATIONS



Three kinds of proof obligations need to be discharged:

1. **Correspondence Theorems**
2. **Preservation Theorems**
3. **Guard Theorems**

PROOF OBLIGATIONS



Three kinds of proof obligations need to be discharged:

1. **Correspondence Theorems**
2. **Preservation Theorems**
3. **Guard Theorems**

Important: ACL2 prints these proof obligations for the user.

Outline

INTRODUCING ABSTRACT STOBJS IN ACL2

1. Introduce a concrete stobj.
2. Define functions that operate on the fields of the concrete stobj (`:EXEC` functions).
3. Define a recognizer function and a creator function for the abstract stobj.
4. Define the functions that will operate on the abstract stobj (`:LOGIC` functions).
5. Define the *correspondence* predicate.
6. Discharge the proof obligations needed to introduce a `defabsstobj` event.
7. Introduce the `defabsstobj` event into ACL2.

INTRODUCING ABSTRACT STOBJS IN ACL2

1. Introduce a concrete stobj.

INTRODUCING ABSTRACT STOBJS IN ACL2

2. Define functions that operate on the fields of the concrete stobj (`:EXEC` functions).

```
(defun !mem$ci (i v x86-32$c)
  ;; Declare statement elided.
  (let* ((i-top (ash i (- *2^x-byte-pseudo-page*)))
         (addr (mem-tablei i-top x86-32$c)))
    (mv-let (addr x86-32$c)
            (cond ((eql addr 1)  ; page is not present
                   (add-page-x86-32$c i-top x86-32$c))
                  (t (mv addr x86-32$c)))
            (!mem-arrayi (logior addr (logand *2^24-1* i))
             v
             x86-32$c))))
```

INTRODUCING ABSTRACT STOBJS IN ACL2

3. Define a recognizer function and a creator function for the abstract stobj.

We choose a sparse alternative representation for the y86 memory: *records*. (`books/defexec/other-apps/records`)

INTRODUCING ABSTRACT STOBJS IN ACL2

4. Define the functions that will operate on the abstract stobj (:LOGIC functions).

```
(defun !mem$ai (i v x86-32)
   ;; Declare statement elided.
  (update-nth *memi*
              (s i v (nth *memi* x86-32))
              x86-32))
```

INTRODUCING ABSTRACT STOBJS IN ACL2

5. Define the *correspondence* predicate.

```
(defun corr (conc abs)
  ;; Declare statement elided.
  (and (x86-32$cp conc)
       (x86-32$ap abs)
       (equal (nth *rgfi* conc) (nth *rgfi* abs))
       (equal (nth *flg* conc) (nth *flg* abs))
       ...
       (corr-mem conc (nth *memi* abs))))
```

Introducing Abstract Stobjs in ACL2

6. Discharge the proof obligations needed to introduce a `defabsstobj` event.

Typically, just execute the `defabsstobj` event, paste the resulting proof obligations (`defthm` events) into your file, and prove them in the normal way.

INTRODUCING ABSTRACT STOBJS IN ACL2

7. Introduce the `defabsstobj` event into ACL2.

INTRODUCING ABSTRACT STOBJS IN ACL2

```
(defabsstobj x86-32
  :concrete x86-32$c
  :recognizer (x86-32p :logic x86-32$ap :exec x86-32$cp-pre)
  :creator (create-x86-32 :logic create-x86-32$a
                          :exec  create-x86-32$c)
  :corr-fn corr
  :exports ((rgfi  :logic rgf$ai  :exec rgf$ci)
            (!rgfi :logic !rgf$ai :exec !rgf$ci)
            (rip   :logic rip$a   :exec rip$c)
            (!rip  :logic !rip$a  :exec !rip$c)
            (flg   :logic flg$a   :exec flg$c)
            (!flg  :logic !flg$a  :exec !flg$c)
                         ...
            (memi  :logic mem$ai  :exec mem$ci)
            (!memi :logic !mem$ai :exec !mem$ci
             :protect t)))
```

Introducing Abstract Stobjs in ACL2

1. Introduce a concrete stobj.
2. Define functions that operate on the fields of the concrete stobj (`:EXEC` functions).
3. Define a recognizer function and a creator function for the abstract stobj.
4. Define the functions that will operate on the abstract stobj (`:LOGIC` functions).
5. Define the *correspondence* predicate.
6. Discharge the proof obligations needed to introduce a `defabsstobj` event.
7. Introduce the `defabsstobj` event into ACL2.

OUTLINE

Outline

SIMPLIFIED REASONING

Memory read-over-write theorem:

```
(defthm read-write
  (implies (and (x86-32$cp x86-32$c)
                (integerp i)
                (<= 0 i)
                (< i *mem-size-in-bytes*)
                (integerp j)
                (<= 0 j)
                (< j *mem-size-in-bytes*)
                (n08p v))
           (equal (mem$ci j (!mem$ci i v x86-32$c))
                  (if (equal i j)
                      v
                      (mem$ci j x86-32$c)))))
```

SIMPLIFIED REASONING

Memory read-over-write theorem without abstract stobjs:

```
(defthm read-write
  (implies (and (x86-32$cp x86-32$c)
                (integerp i)
                (<= 0 i)
                (< i *mem-size-in-bytes*)
                (integerp j)
                (<= 0 j)
                (< j *mem-size-in-bytes*)
                (n08p v))
           (equal (mem$ci j (!mem$ci i v x86-32$c))
                  (if (equal i j)
                      v
                      (mem$ci j x86-32$c)))))
```

SIMPLIFIED REASONING: ELIMINATING HYPOTHESES

Abstract stobjs allow us to prove the following read-over-write theorem instead:

```
(defthm memi-!memi
  (equal (memi i (!memi j v x86-32))
         (if (equal i j)
             (or v 0)
             (memi i x86-32))))
```

SIMPLIFIED REASONING: ELIMINATING HYPOTHESES

Abstract stobjs allow us to prove the following read-over-write theorem instead:

```
(defthm memi-!memi
  (equal (memi i (!memi j v x86-32))
         (if (equal i j)
             (or v 0)
             (memi i x86-32))))
```

Removing hypotheses from theorems:

- results in stronger (more general) rules, and

SIMPLIFIED REASONING: ELIMINATING HYPOTHESES

Abstract stobjs allow us to prove the following read-over-write theorem instead:

```
(defthm memi-!memi
  (equal (memi i (!memi j v x86-32))
         (if (equal i j)
             (or v 0)
             (memi i x86-32))))
```

Removing hypotheses from theorems:

- results in stronger (more general) rules, and
- speeds up the ACL2 rewriter during proofs.

OUTLINE

EXECUTION IN ACL2

- We use *guards* extensively to do processor modeling.

EXECUTION IN ACL2

- We use *guards* extensively to do processor modeling.
- While executing functions, ACL2 uses guards to check the legality of each call made in the ACL2 loop.

Execution in ACL2

- We use *guards* extensively to do processor modeling.
- While executing functions, ACL2 uses guards to check the legality of each call made in the ACL2 loop.
- All functions that take the concrete stobj as input have `x86-32$cp` as a guard.

EXECUTION IN ACL2

- We use *guards* extensively to do processor modeling.
- While executing functions, ACL2 uses guards to check the legality of each call made in the ACL2 loop.
- All functions that take the concrete stobj as input have `x86-32$cp` as a guard.
- `x86-32$cp` is a complicated predicate that makes guard-checking slow.

EXECUTION IN ACL2

- We use *guards* extensively to do processor modeling.
- While executing functions, ACL2 uses guards to check the legality of each call made in the ACL2 loop.
- All functions that take the concrete stobj as input have `x86-32$cp` as a guard.
- `x86-32$cp` is a complicated predicate that makes guard-checking slow.
- In the case of abstract stobjs:
  - We prove that the recognizer always holds for the abstract stobj returned by the updater functions.

EXECUTION IN ACL2

- We use *guards* extensively to do processor modeling.
- While executing functions, ACL2 uses guards to check the legality of each call made in the ACL2 loop.
- All functions that take the concrete stobj as input have `x86-32$cp` as a guard.
- `x86-32$cp` is a complicated predicate that makes guard-checking slow.
- In the case of abstract stobjs:
  - We prove that the recognizer always holds for the abstract stobj returned by the updater functions.
  - **Optimization: calls of `x86-32p` evaluate instantly to `T`.**

OUTLINE

PROOF BY SYMBOLIC EXECUTION

- All elements of `x86-32$c` are logically represented as linear lists.

PROOF BY SYMBOLIC EXECUTION

- All elements of `x86-32$c` are logically represented as linear lists.
- Concrete stobjs give us an enormous logical representation of the state because of the large memory arrays.

PROOF BY SYMBOLIC EXECUTION

- All elements of `x86-32$c` are logically represented as linear lists.
- Concrete stobjs give us an enormous logical representation of the state because of the large memory arrays.
- Every read/write operation involving `x86-32$c` requires linear traversals.

PROOF BY SYMBOLIC EXECUTION

- All elements of `x86-32$c` are logically represented as linear lists.
- Concrete stobjs give us an enormous logical representation of the state because of the large memory arrays.
- Every read/write operation involving `x86-32$c` requires linear traversals.
- Instead, we use sparse data structures — *records* — to model the memory.

PROOF BY SYMBOLIC EXECUTION

- All elements of `x86-32$c` are logically represented as linear lists.
- Concrete stobjs give us an enormous logical representation of the state because of the large memory arrays.
- Every read/write operation involving `x86-32$c` requires linear traversals.
- Instead, we use sparse data structures — *records* — to model the memory.
- Initial representation of memory is now `NIL`, as opposed to large lists of zeroes in the concrete stobj representation.

PROOF BY SYMBOLIC EXECUTION

- We get a smaller processor state that is amenable to proof by symbolic execution.

PROOF BY SYMBOLIC EXECUTION

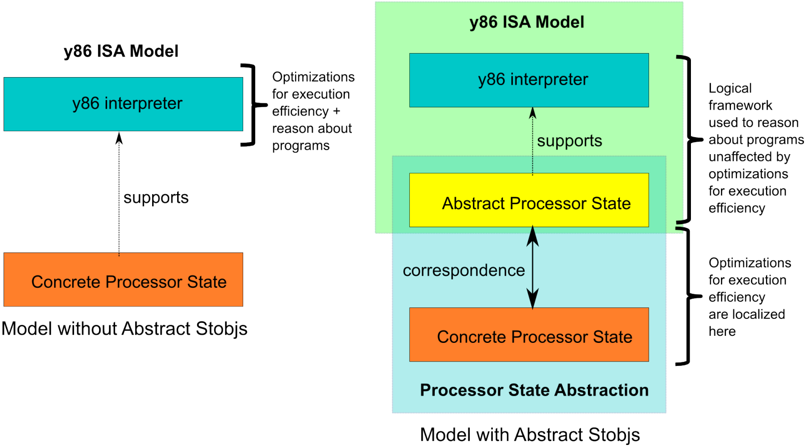
- We get a smaller processor state that is amenable to proof by symbolic execution.
- Abstract stobjs enable the use of `GL` (`books/centaur/gl`) to do automatic proofs about some non-trivial y86 binary programs.

PROOF BY SYMBOLIC EXECUTION

- We get a smaller processor state that is amenable to proof by symbolic execution.
- Abstract stobjs enable the use of `GL` (`books/centaur/gl`) to do automatic proofs about some non-trivial y86 binary programs.
- For an example, see `books/models/y86/y86-two-level-abs/examples`.

Outline

Layered Modeling Strategy



Abstract stobjs avoid the need of a trade-off between reasoning and execution efficiency.

OUTLINE

CONCLUSION

In brief, abstract stobjs:

- make processor modeling easier by **introducing abstraction** — a layered model is more manageable and robust;

CONCLUSION

In brief, abstract stobjs:

- make processor modeling easier by **introducing abstraction** — a layered model is more manageable and robust;
- make reasoning about big models easier through **elimination of hypotheses**; and

CONCLUSION

In brief, abstract stobjs:

- make processor modeling easier by **introducing abstraction** — a layered model is more manageable and robust;
- make reasoning about big models easier through **elimination of hypotheses**; and
- support **efficiency of concrete execution** (with faster guard checking) and **symbolic execution** (by presenting sparse structures like records instead of long lists).

See :DOC defabsstobj.

Abstract Stobjs and Their Application to ISA Modeling

Shilpi Goel
Warren A. Hunt, Jr.
Matt Kaufmann

The University of Texas at Austin

30^{th} May, 2013