Extended Abstract: Mutable Objects with Several Implementations

Matt Kaufmann (speaker)
Yahya Sohail
Warren A. Hunt, Jr.

UT Austin

May 12, 2025

YET ANOTHER STOBJ DEVELOPMENT

- Single-threaded objects, or *stobjs*, support efficient execution.
- These are mutable objects with syntactic restrictions that allow for a purely functional semantics.
- This talk will try not to assume experience with stobjs. For more background:
  - See :DOC stobj about *concrete* stobjs; and
  - see :DOC defabsstobj about *abstract* stobjs.

Also see :DOC attach-stobj.

MOTIVATION

The x86 model has been used to boot Linux and run Linux jobs.

- Different applications can perform best with different memory models.
- One solution might be to edit the x86 books to include different memory models. But:
  - we may need to prove the same theorem repeatedly for different memory models (e.g., read-over-write).
- It would be great to have one *logical* memory model supported by different *executions*.

Of course, applications other than x86 could have similar issues.

EXAMPLE

100,000 writes to a memory model.

Memory	Benchmark	Time (secs)	Size (bytes)
symmetric	low	2.75	2000085072
symmetric	high	2.75	2000085072
asymmetric	low	0.00	6663495760
asymmetric	high	87.91	6666641488
attached	low	0.00	6899818576
attached	high	89.04	6902964304

```
----- symmetric: -----
(include-book  "centaur/bigmems/bigmem/bigmem" :dir :system)
----- asymmetric: -----
(include-book  "centaur/bigmems/bigmem-asymmetric/bigmem-asymmetric"
               :dir :system)
----- attached: -----
(include-book  "centaur/bigmems/bigmem-asymmetric/bigmem-asymmetric"
               :dir :system)
(attach-stobj bigmem::mem bigmem-asymmetric::mem)
(include-book  "centaur/bigmems/bigmem/bigmem" :dir :system)
```

symmetric: Include "bigmem"

---

asymmetric: Include "bigmem-asymmetric"

---

attached:
Include "bigmem-asymmetric"
(attach-stobj bigmem::mem bigmem-asymmetric::mem)
Include "bigmem"

```
(in-package  "BIGMEM-ASYMMETRIC")
(acl2::defabsstobj mem ...
  :exports ((read-mem  :logic read-mem$a ...)
            (write-mem :logic write-mem$a ...)) ...)

(in-package  "BIGMEM")
(acl2::defabsstobj mem ...
  :attachable t ...
  :exports ((read-mem  :logic read-mem$a ...)
            (write-mem :logic write-mem$a ...)))
```

BENEFITS OF `ATTACH-STOBJ`

- Saves proof work:
  Several models can be used without replicating proofs.
  - Theorems (e.g., read-over-write) are proved only for the attachable stobj.
- Saves certification and replication:
  A single book can use several models for execution.
  - Certify `gen.lisp`, which introduces an attachable (generic) stobj, `gen`.
  - Certify an application book, `app.lisp`, that includes `gen.lisp`.
  - Now we can run that application with different implementations:
    - include a book $\texttt{impl}_i\texttt{.lisp}$ introducing an implementation stobj $\texttt{impl}_i$ together with $\texttt{(attach-stobj gen impl}_i\texttt{)}$; then
    - include `app.lisp`.
- The performance hit is minor.

AN IMPLEMENTATION CHALLENGE

Suppose `app.lisp` includes `gen.lisp` but not `impl.lisp`.

Certify all books and then evaluate the following sequence of events.

1. `(include-book "impl")` ; defines implementation stobj `impl`
2. `(attach-stobj gen impl)`
3. `(include-book "gen")` ; defines attachable stobj `gen`
4. `(include-book "app")` ; defines function `foo`

Also suppose we have the following.

- `(defabsstobj gen .. :exports ((E .. :exec` E_{gen}`)))`
- `(defabsstobj impl .. :exports ((.. :exec` E_{impl}`)))`
- `(defun foo (gen) (declare ..) (E gen))` ; from app.lisp

`E` is a macro, so the compiled code for `foo` from `"app"` calls E_{gen}.

But after #1-4 above (note `attach-stobj` call), `foo` should call E_{impl}.

Solution: ACL2 tracks functions like `foo` and compiles them while including the book (ignoring the compiled code from certification).

FURTHER READING

See :DOC `attach-stobj` for usage details.

See community books directory `demos/attach-stobj/` for a worked example, starting with file `README.txt`.

Performance testing (discussed on preceding slides) is in subdirectory `mem-test/` of that directory.

And of course, see the paper, which in particular discusses:

- the use of keyword argument `:non-executable t` to save space; and
- some tricky implementation issues.

For details, see the 664-line "Essay on Attachable Stobjs" in ACL2 source file `basis-b.lisp`.

CONCLUSION

- A key design goal for ACL2 is for it to serve as a programming language that executes efficiently.
- Stobjs provide significant support for efficient execution.
- *Abstract stobjs* and *nested stobjs* provide added flexibility.
  - Example: x86 model
- `Attach-stobj` is another step towards flexible support for efficient execution.

We thank:

- Sol Swords for helpful design feedback;
- ForrestHunt, Inc. for supporting that implementation;
- the reviewers for helpful feedback on this paper; and
- ***you***, for listening.