Polymorphism in ACL2

Ruben Gamboa
University of Wyoming
ACL2 Workshop 2003

Motivation

- *Preaching to choir: ACL2 is a terrific theorem prover that has been used successfully in several verification efforts*
- *We are interested in verifying software*
- *Objects are today's dominant software metaphor*
- *ACL2 does not support objects (yet)*

Supporting Objects

- *Objects encapsulate both state and behavior*
- *To support objects, we need*
  - *Support for object state, e.g. defstructure or stobjs*
  - *Support for polymorphism*

Object State

- *Differentiate between two notions*
  - *An object has state*
  - *A reference points to an object*
- *There is only one copy of any object*
- *There can be many references to the same object*

Object State

- *Objects live in the stobj* `memory`
  - *ensures there is only one copy*
  - *allows efficient access to object*
- *References are regular ACL2 elements*
  - *e.g.,* `(cons 'square 18)` *points to the object (which must be a* `square`*) at location 18 of* `memory`

The `memory` Stobj

- *The* `memory` *stobj plays a central role*
- *All functions referencing objects must include it in their signature*
- *Subject to the usual restrictions on usage of stobjs*
- *References, however, are not subject to these restrictions*

Polymorphism

- *Our focus is polymorphism, a cornerstone of object-oriented programming*
- *Polymorphism allows a caller to send a message to an object (i.e., call a function) without knowing the exact type of the object*
- *Used in inheritance and with interfaces*

Introducing Classes

```
(defclass measurable nil

  ((v :type integer :initially 0))

  (defmethod measure (x memory)
    (abs (measurable..v x memory)))

  (defthm measure-is-non-negative-real
    (implies (measurable-p x)
             (and (realp (measure x memory))
                  (<= 0 (measure x memory)))))

 ...)
```

Defining Axioms

- `(measurable-p x)`
- `(strict-measurable-p x)`
- `(comparable..new memory)`
- `(measurable..v x memory)`
- `(measurable..update-v x new-v memory)`

Note: `measurable-p` *does not imply* `strict-measurable-p`

Method Definitions

- *Definition:*
  - ```
    (implies (strict-measurable-p x)
             (equal (measure x memory)
                    (abs (measurable..v x memory))))
    ```
-
- *Constraint:*
  - ```
    (implies (measurable-p x)
             (and (realp (measure x memory))
                  (<= 0 (measure x memory))))
    ```

The Role of Constraints

- *Before a* defclass *event is accepted, ACL2 verifies that all the constraints are valid for objects of this specific class*
- *This also applies to any constraints set by ancestor classes*
- *Similar to constraints in an* encapsulate *event -- but methods are executable*

Subclassing

```
(defclass complex measurable
  ((a :type real :initially 0)
   (b :type real :initially 0))

  (defmethod measure (c memory)
    (let ((x (complex..a c memory))
          (y (complex..b c memory)))
      (acl2-sqrt (+ (* x x) (* y y)))))
  ...)
```

Subclassing and Constraints

- *Before the defclass is accepted, ACL2 verifies the following obligation*
  - ```
    (implies (strict-complex-p x)
             (and (realp (measure x memory))
                  (<= 0 (measure x memory))))
    ```
- *Notice* `measurable-p` *has been replaced by* `strict-complex-p` *in the hypothesis*
    ```

Subclassing and Definitions

- *New definition:*

```
(implies (or (strict-measurable-p x)
             (strict-complex-p x))
         (equal (measure x memory)
                (cond ((strict-measurable-p x)
                       (abs (measurable..v x memory)))
                      ((strict-complex-p x)
                       (let ((a (complex..a x memory))
                             (b (complex..b x memory)))
                         (acl2-sqrt (+ (* a a) (* b b))))))))
```

Soundness

- *"New definition" sounds like it opens a door to* nil
- *But redefinitions are very restricted*
  - *Essentially, methods are functions defined in a major case-split*
  - *Subclasses add cases to the split, but they never change or delete old cases*

Soundness

- *At any given time, we know of only some possible subclasses of a class*
- *But for each class we can add an unspecified "other" predicate to complete the definition*
- *The complete definition is valid in ACL2*

Soundness

- *The complete definition implies all of the "partial" definitions*
- *If we choose the "other" case carefully, it also satisfies all the constraints*
- *Thus, we can replace* defclass *with a complete definition of the methods, followed with proof of the partial definitions*

Soundness: The Translator

- *The argument can be formalized as a translation from ACL2+defclass histories into regular ACL2 histories*
- *We use this translation to define the semantics of ACL2+defclass*

Inheriting Functions

- *Consider a list of references to* measurable *objects*
- *It is easy to define a* max-measure *function that finds the maximum* measure *in the list*
- *This function will also work on lists of* complex-p *objects -- implicitly*
- *Theorems about this function will also apply to complex-p objects -- implicitly*

Current Status

- *We have a "working" translator*
- *Translations require post-processing by hand*
- *We have verified some "toy" problems*

Future Work

- *Verify something more substantial*
  - *e.g., an abstract hashtable, a concrete red-black tree, and a user of hashtables, like a BDD translator*
- *Modify ACL2 to support* defclass *natively*