Pythia: Automatic Generation of Counterexamples for ACL2 using Alloy

Alexander Spiridonov
Sarfraz Khurshid

Agenda

- Making ACL2 more novice-friendly
- Pythia: using Alloy to find counterexamples
- An end-to-end example
- Modeling ACL2 objects in Alloy
- Discussion and future work
- Q & A

The Problem

- Challenge for ACL2 community: making ACL2 more accessible to novices
- Pain point: failed proof attempts
  - The formula may be beyond the prover's reach
  - The formula may not be a theorem
- Solution: counterexamples
  - Illustrate why the proof attempt has failed
  - Suggest a way to turn the formula into a theorem
- How to generate counterexamples automatically?

Our solution: Pythia

- We use Alloy to find counterexamples to ACL2 non-theorems:
  1. Model ACL2 objects in the Alloy specification language
  2. Use the Alloy Analyzer to generate an instance of the model
  3. Translate the instance into ACL2 objects
  4. Evaluate the formula on these objects
  5. Report counterexamples

The Alloy language

- Strongly-typed, first-order declarative language based on sets and relations
- An Alloy model is comprised of modules, which contain signatures and constraints
- *Signatures* define the objects in the model and the relations between them

The Alloy language

- An Alloy model of a binary tree:

```
sig Node {
  left, right: lone Node,
  value : lone Int
}

sig Tree {
  root: Node
}
```

The Alloy language

- *Constraint paragraphs* include facts and predicates
  - *Facts* are constraints that always hold
  - *Predicates* are named constraints

```
pred Acyclic (t: Tree) {
  all n : t.root.*(left+right) |
    n !in n.^(left+right)
}
```

The Alloy Analyzer

- An automatic, SAT-based analyzer that generates instances of Alloy models
- The Analyzer limits its search to a finite scope
- Commands tell the Analyzer to find instances of the model that satisfy the predicate’s constraints:

```
run Acyclic for 3 but 1 Tree
```

The Alloy Analyzer

- A sample instance:

```
Int = {0, 1, 2, 3}
Tree = {T0}
Node = {N0, N1, N2}
// fields of Tree
root = {(T0, N0)}
// fields of Node
left = {(N0, N1)}
right = {(N0, N2)}
value = {(N0, 2), (N1, 1), (N2, 3)}
```

- Alloy visualizer for displaying instances

An Example

```
(defun orderedp (x)
   (cond ((endp x) t))
         ((endp (cdr x)) t))
         (t (and (<= (car x) (cdr x)) (orderedp (cdr x))))))

(defun ordered-list-of-intsp (x)
   (and (integer-listp x) (orderedp x)))

(defun my-merge (x y)
   (declare (xargs :measure (+ (len x) (len y))))
   (if (and (consp x) (consp y))
         (cond ((< car x) (car y)) (cons (car x) (my-merge (cdr x) y)))
               ((> car x) (car y)) (cons (car y) (my-merge  x (cdr y))))
               (t (cons (car x) (my-merge (cdr x) (cdr y))))))
     (if (endp x)
         y
       x)))
```

An Example

- Suppose the user attempts to prove the following:

```
(defthm properties-of-my-merge
  (implies (and (ordered-list-of-intsp x)
                (ordered-list-of-intsp y))
           (and (orderedp (my-merge x y))
                (equal (len (my-merge x y))
                       (+ (len x) (len y))))))
```

- The proof attempt will fail, but it may be hard for a novice user to figure out why

An Alloy model of ACL2 objects

- A cons tree:

```
pred Cons(t: Tree) {
  all n : t.root.*(left+right) |
      n !in n.^(left+right)
  all n : t.root.*(left+right) |
      lone n.~(left+right)
  all n : t.root.*(left+right) |
      some n.(left+right) => no n.value
  all n : t.root.*(left+right) |
      no n.(left+right) or
      #n.(left+right) = 2
}
```

An Alloy model of ACL2 objects

- A proper cons tree and a true list of atoms:

```
pred ProperCons(t: Tree) {
  Cons(t)
  one n : t.root.^right |
      no n.(left+right) && no n.value
}

pred TrueListOfAtoms(t: Tree) {
  ProperCons(t)
  all n : t.root.*(left+right) |
      some n.(left+right) =>
             one n.left.value && no n.right.value
}
```

An Alloy model of ACL2 objects

- An ordered list of integers:

```
pred Ordered(t: Tree) {
all n : t.root.*(left+right) |
  all v : n.left.*(left+right).value |
      all w : n.right.*(left+right).value |
            int v < int w
}

pred OrderedListOfIntegers(t: Tree) {
  TrueListOfAtoms(t)
  Ordered(t)
}
```

An Alloy model of ACL2 objects

From ACL2…

```
(defun ordered-list-of-intsp (x)
  (and (integer-listp x) (orderedp x)))
```

…to Alloy:

```
pred OrderedListOfIntegers(t: Tree) {
  TrueListOfAtoms(t)
  Ordered(t)
}
```

Finding a counterexample

- Alloy Analyzer generates an instance of the model that satisfies the predicate `OrderedListOfIntegers` within the specified scope (user-adjustable)
- Pythia translates the instance into a set of ACL2 objects, e. g. `‘(1 2 3), ‘(-1 0 1), ‘(3 4 5)`
- ACL2 evaluates the formula on this set of objects and reports a counterexample: `x = ‘(1 2 3)`, `y = ‘(3 4 5)`

Effectiveness

- Pythia works well for classic ACL2 non-theorems such as `(equal (rev (rev x)) x)`
- Alloy predicates that can be reused in other, more complex models
- As formulas become more complex, it becomes increasingly difficult and error-prone to construct potential counterexamples by hand, and a mechanical tool becomes more useful
  - Example: JVM state in the ACL2 JVM model

Limitations and future work

- In its present form, Pythia has several limitations:
  - Analysis is incomplete
    - Can increase analyzer scope
  - User has to write Alloy models for more complex ACL2 definitions
    - Alloy does not support recursive definitions, but workarounds exist
- Future work: automatically translate ACL2 definitions into Alloy

Conclusion

- Challenge for ACL2 community: Making ACL2 more accessible to novices
- Automatically generating counterexamples for ACL2 non-theorems could help novices and serve as an educational tool
- By using the Alloy language and analyzer, Pythia automatically finds counterexamples for the kinds of non-theorems novices are likely to encounter
- An ACL2-Alloy translator would enable Pythia to tackle more complex formulas

Q & A