relational invariants

Daniel Jackson

november 14, 2007
topics for today

textual invariants
  • navigations and quantifications

examples from Java library
  • rep invariants of LinkedList and HashMap

relational invariants as code annotations
  • conventions for writing invariants about code objects
  • modifies and effects clauses

  • EXIF filtering application
textual relational invariants
example: CSAIL accounts

why more expressiveness is needed
• ... and why role based access control isn’t good enough

CSAIL uses the following model for setting up accounts
• new lab member applies for account with name of research group
• group leader or her assistant can approve
• can’t express this in diagram alone
here's how we can write the constraint textually

\[
\text{all } u_1, u_2: \text{User} \mid u_2 \in u_1.\text{approves} \implies \\
\text{some } g: \text{Group} \mid \\
\quad u_2 \in g.\text{members} \text{ and } \\
\quad \text{some } f: g.\text{leaders} \mid u_1 = f \text{ or } u_1 = f.\text{assistant}
\]

that is:

user u1 can approve user u2 if u1 is the leader of, or assistant of a leader of, a group that u2 is a member of
constraint notation

that constraint is in Alloy

\* a relational constraint notation developed here at MIT
\* for this class, don’t need to know details of notation
\* but should understand basic form, common to object modelling languages

syntactic elements

\* **navigation expressions**: define sets of objects
  
  eg, `g.members` // set of members of group `g`

\* **basic constraints**: formed by comparing navigation expressions
  
  eg, `u2 in g.members` // `u2` is a member of group `g`

\* **quantifications**: over basic constraints
  
  eg, `some g: Group | u2 in g.members` // there’s some group `u2` is a member of
idea

expression represents one or more paths in the object graph

operators

s.r denotes the set of objects you get to following relation r once, starting from set of objects s

s.*r set of objects you get to following r zero or more times from s

s.^r set of objects you get to following r one or more times from s

s.~r set of objects you get to following r once, backwards, from s

s + t set of objects you get to following s or t

s & t set of objects you get to following s and t

s - t set of objects you get to following s but not t

v scalar variable, denotes the set \{v\}
write navigation expressions for

- the assistants of the leaders of group g
- the users who are members of both groups g1 and g2
- the set of staff who have approved one or more users
basic constraints

idea

• just compare two navigation expressions as sets

operators

\[ s = t \quad \text{s and t denote the same set of objects} \]
\[ s \neq t \quad \text{s and t denote different sets of objects} \]
\[ s \text{ in } t \quad \text{s denotes a subset of t} \]

simple compound constraints with

and, or, not

write basic constraints for

• the leaders of group g are members of group g
• the assistant of faculty f belongs to the same group as f
quantifications

idea
• take a basic constraint with a free variable
• apply it to all bindings of the free variable to members of some set

forms

\[ \text{all } v: s | F \] \quad F \text{ holds for all values of } v \text{ in the set } s

\[ \text{some } v: s | F \] \quad F \text{ holds for some value of } v \text{ in the set } s

\[ \text{no } v: s | F \] \quad F \text{ holds for no value of } v \text{ in the set } s

\[ \text{one } v: s | F \] \quad F \text{ holds for one value of } v \text{ in the set } s

also

\[ \text{some } s \] \quad s \text{ contains some elements}

\[ \text{no } s \] \quad s \text{ contains no elements}
write constraints for

• every group leader is a member of that group
• every faculty who leads a group with some members has an assistant
summary: textual constraints

a simple and expressive notation
\* we’ve seen it for **invariants**
\* but can also be used to describe **operations**

access-control
\* current research* investigating languages like this

model-driven architecture
\* generate code automatically from object models
\* research and industrial projects aiming to do this

Java library: LinkedList
Java library's linked list

from Java LinkedList:

```java
class LinkedList { Entry header; ...}
class Entry { Object element; Entry prev; Entry next; ...}
```

rep invariants: can you read these?

- `header != null`
- `header in header.^next`
- `all e: header.*next | e.next != null and e.prev != null`
- `all e: header.*next | e.next.prev = e`

how do these help?

- non-null: don’t need lots of checks
- cycle: no special treatment for start/end
- prev/next pairs: easy to insert and delete entries

exercise: draw graphical OM from decls
abstract fields and specs
abstract fields

consider writing a spec of Set.add

```java
class Set { void add (Object o); ...}
class ListSet implements Set { List elementList; ...}
class LinkedList { Entry header; ...}
class Entry { Object element; Entry prev; Entry next; ...}
```

write it in terms of `elementList.header.*next.element` etc?

' abstraction violation!

better way to write spec

' ‘specification’ or ‘abstract’ field `elements` whose type is a mathematical set

```java
@specfield _Set elements;
public void add (Object o)
@requires true
@modifies this.elements
@effects this.elements’ = this.elements + o
```
spec structure

sample spec

```java
@specfield _Set elements;
public void add (Object o)
@requires true
@modifies this.elements
@effects this.elements' = this.elements + o
```

clauses

\* requires: precondition, as before
\* modifies: lists expressions denoting fields that might change
\* effects: describes effects of execution on fields, with e’ for e after
more accurate spec of Set.add

real spec is nearer to this

@specfield _Set elements;
public void add (Object o)
@modifies this.elements
@throws NullPointerException
@effects
  if (o == null) and (null in elements) or (some e: elements | e.equals(o))
    elements' = elements
  else if (o != null)
    elements' = elements + o
  else
    (elements' = elements + o) or (throws NullPointerException)

actually, even worse

- can throw ClassCastException if set can’t contain elements of that type
mathematical abstract fields
- \_Set\langle E \rangle : set of objects of type E
- \_Map\langle K, V \rangle : maps each key of type K to at most one value of type V
- \_Relation\langle K, V \rangle : maps each key of type K to zero or more values of type V

using a relation-valued field
- k.(o.map) is image of key k under map field of object o

using observer methods
- can use observer methods in expressions, eg. x.equals(y), m.map.keys()

modifies syntax
- if any field of object o can be modified, write modifies o.*
Java library: HashMap
Java library's hash table

from Java HashMap:

```java
class HashMap<K,V> { Entry[] table; }
class Entry<K,V> { K key; V value; Entry<K,V> next; }
```

exercise

- given explanation of how hash map works, write rep invariants

essential invariant

```java
all i: int, k: Key | k in table[i].next.key implies k.hashCode() = i
```

question

if k1 has been inserted, and k1.equals(k2), what must the relationship be between k1.hashCode() and k2.hashCode() for k1 to be found when k2 is looked up?
example: EXIF filtering
conceptual model of caching

notes

• some subset of filters are cached, and have matches stored against them
• each of these cached filters subsumes some set of filters
code model of caching

ideas
- FilterTree implements subsumes
- TrueFilter allows more uniform coding of FilterTree
- FilterMap holds cached query results against filters

limitations
- only basic filters cached
- only used at top level of filter
public class FilterTree {
/**
 * @specfield _Relation<Filter, Filter> subsumers;
 * @specfield _Set<Filter> filters;
 *
 * Abstract invariant:
 * // the subsumers relation correctly relates the members of the set filters
 * all f, f': filters | f' in f.subsumers iff f'.subsumes(f) and !f.equals(f')
 *
 * Abstraction function:
 * // set of filters is the parent filter and the union of the sets represented by the children
 * filters = filter + children.elements.filters
 *
 * // subsumers relation is just that defined by the filters themselves
 * subsumers = {f, f': filters | f'.subsumes(f) and !f.equals(f')}
 *
 * Rep invariant:
 * // parent subsumes each of its children
 * all cf: children.filter | filter.subsumes(cf)
 *
 * // no other filter comes between child and parent
 * all cf: children.filter | no f: filters | f.subsumes(cf) and filter.subsumes(f)
 * and !f.equals(cf) and !f.equals(filter)
 *
 * // all fields non null
 */

Filter filter;
List<FilterTree> children;

---

things to note:
-- data structure only holds part of abstract model
-- rep invariant explains
public class Catalog {

/**
 * Mutable object representing a set of photos and cached subsets corresponding
 * to the results of running some filters.
 *
 * // the set of photos stored
 * @specfield Set<Photo> photos;
 *
 * Abstraction function:
 * // identity on set of photos stored
 *
 * Rep invariant:
 * // all photos in cached matching sets are stored in the catalog
 * all f: Filter | f.filterToMatches in photos
 *
 * // the filter in the tree are those with a cached matching set
 * filterToMatches.keys() = tree.filters
 *
 * // all fields non null
 */

Set<Photo> photos;
FilterTree tree;
Map<Filter, Set<Photo>> filterToMatches;
bug in Catalog code

what's wrong with this code?

‣ why did I initially write it this way?
‣ how should it be fixed?

/**
 * Register a filter and cache its results
 *
 * @requires filter != null
 *
 */

public void registerFilter (Filter filter) {
    if (!(filter instanceof BasicFilter))
        return;
    tree.add (filter);
    Set<Photo> matches = getMatches (filter);
    filterToMatches.put(filter, matches);
}
summary

textual constraints

› are state machine invariants!
› simple but expressive notation
› more succinct than code (consider, eg. `header.*next.elements`)

abstract fields and specs

› view heap in terms of specification not implementation fields
› modifies and effects clauses

Java library and EXIF examples

› show how rep invariants especially are fundamental to design