Julien Brunel, David Chemouil, Alcino Cunha, Eunsuk Kang, Nuno Macedo

FORMAL SOFTWARE DESIGN WITH ALLOY AND ELECTRUM

ANALYSIS

Universidade do Minho & INESC TEC

ONERA DTIS & Université fédérale de Toulouse

Carnegie Mellon University

3rd World Congress on Formal Methods, Porto, Portugal, October 2019

OPERATIONS

OPERATIONS AS PREDICATES

Back to: Simple social network

```
sig SocialNetwork {
  friends : User -> User,
  posts : User -> Post
}

pred addPost[n1, n2 : SocialNetwork, u : User, p : Post] {
    // precondition
    // post condition
    n2.posts = n1.posts + u -> p
    // frame condition
    n2.friends = n1.friends
}
```

SIMULATING AN OPERATION

```
run executeAdd {
    some n1, n2 : SocialNetwork, u : User, p : Post] |
        addPost[n1,n2,u,p]
}
```

SCOPES

```
run executeAdd {
    some n1, n2 : SocialNetwork, u : User, p : Post] |
        addPost[n1,n2,u,p]
} for 5 but 3 User, exactly 2 SocialNetwork
```

ANALYSIS

ANALYSIS IN ALLOY

- Check:
  - Consistency of a specification
  - Invariants
  - Algebraic properties of operations
  - Refinement
  - Temporal properties

INVARIANT CHECK

```
check addPostPreservesInvariant {
    all n1, n2 : SocialNetwork, u : User, p : Post] |
        invariant[n1] and addPost[n1,n2,u,p] implies
            invariant[n2]
}
```

PROPERTY CHECKING AS CONSTRIANT SOLVING

```
check addPostPreservesInvariant {
    all n1, n2 : SocialNetwork, u : User, p : Post] |
        invariant[n1] and addPost[n1,n2,u,p] implies
            invariant[n2]
}
```

is equal to

```
run addPostViolatesInvariant {
    some n1, n2 : SocialNetwork, u : User, p : Post] |
        invariant[n1] and addPost[n1,n2,u,p] and
            invariant[n2]
}
```

CHECKING ALGEBRAIC PROPERTIES

ALGEBRAIC PROPERTIES

If we add a post and immediately delete it, do we end up in the original state?

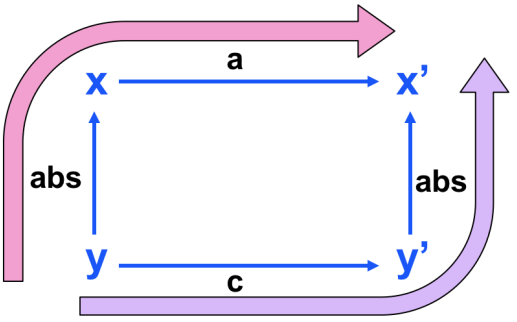
```
check addThenDeleteIsUndo {
    all n1, n2, n3 : SocialNetwork, u : User, p : Post |
        invariant[n1] and
        addPost[n1,n2,u,p] and delPost[n2,n3,u,p] implies
            n1.posts = n3.posts
}
```

REFINEMENT

COMMUTING DIAGRAM

Is the distributed social network a correct implementation of the simple social network?

COMMUTING DIAGRAM



where x = α(y) ; x' = α(y')

ABSTRACTION FUNCTION

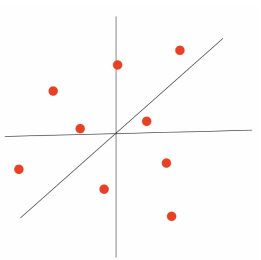
```
fun abs[c : DistributedSN] : SocialNetwork {
    { a : SocialNetwork |
        a.posts = c.servers.posts and
        a.friends = c.friends }
}
```

REFINEMENT CHECK

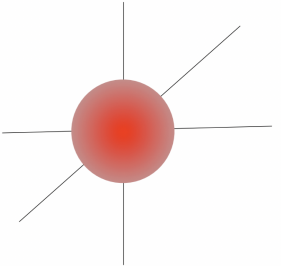
```
assert AddPostRefinement {
    all a1, a2 : SocialNetwork, c1, c2 : DistributedSN,
        u : User, p : Post |
            addPostDistributed[c1, c2, u, p] and
            a1 = abs[c1] and a2 = abs[c2] implies
                addPost[a1, a2, u, p]
}

check AddPostRefinement
```

BRIEF THEORY BEHIND ANALYSIS



Testing
A few instances of arbitrary size



Alloy: Bounded, exhaustive
All instances up to the given scope

- Logic in Alloy is undecidable
- Approach: Fininte Model Finding
  - User provides a scope on each signature
  - Translation to a conjunctive normal form (CNF)
  - Off-the-shelf SAT solvers for instance generation

MODELING IDIOMS IN ALLOY

ENCODING DYNAMIC BEHAVIORS

- Alloy has no built-in notion of states & operations
- Different idioms for encoding states in Alloy
  - A common idiom is to explicitly encode states as objects

```
sig State {
    field : A -> B
}
pred op[s1, s2 : State] { ... }
```

ANOTHER IDIOM: DYNAMIC (OR TIMED) RELATION

```
open util/ordering[Tick]

sig Tick {}
sig Component {
    state : A -> B -> Tick
}
pred step[t1, t2 : Tick, c : Component] { ... }
```

- Add Tick as the last column of mutable relations

SOCIAL NETWORK EXAMPLE

```
sig Tick {}
sig SocialNetwork {
  friends : User -> User -> Tick
  posts : User -> Post -> Tick
}
pred addPost[t1, t2 : Tick, n : SocialNetwork, u : User, p : Post] {
    // post condition
    n.posts.t1 = n.posts.t2 + u -> p
    // frame condition
    n.friends.t1 = n.friends.t2
}
```

TRACES IN ALLOY

```
fact Traces {
    all t1 : Tick - last | let t2 = t1.next |
        all n : SocialNetwork |
            some u : User, p : Post |
                addPost[t1, t2, n, u, p] or
                removePost[t1, t2, n, u, p]
}

run {} for 3 but 5 Tick
```

DYNAMIC RELATION IDIOM: BENEFITS & LIMITATIONS

- (+) Clear separation of static and dynamic parts of the state
- (+) Ability to generate traces and specify temporal assertions
- (-) Limited analysis support (liveness, fairness); bounded
- (-) Can be cumbersome to write

Can we overcome these limitations? Next session: Electrum!

EXERCISES

https://github.com/haslab/Electrum2/wiki/Social-Network

(exercises 3-4)