Alloy

31/03/19

Kalev Alpernas
Let’s Model a Family Tree
Let’s Model a Family Tree

• First, we need to define a domain:
  
  sig Person {}
Let’s Model a Family Tree

• First, we need to define a domain:
  sig Person {}

• Need parents to define a family tree:
  sig Person {
    father: lone Person,
    mother: lone Person
  }
Let’s Model a Family Tree

• First, we need to define a domain:
  sig Person {}  

• Need parents to define a family tree:
  sig Person {
    father: lone Person,
    mother: lone Person
  }

• Alloy gives us some weird families.
Let’s add constraints

• Father and mother are different:

    fact { all p: Person | p.father != p.mother }
Let’s add constraints

• Father and mother are different:
  
  ```
  fact { all p: Person | p.father != p.mother }
  ```

• A person can’t be their father or mother:
  
  ```
  fact { all p: Person | p.father != p and p.mother != p }
  ```
Let’s add constraints

• Father and mother are different:
  \[
  \text{fact} \{ \text{all } p: \text{Person} \mid p.\text{father} \neq p.\text{mother} \}
  \]

• A person can’t be their father or mother:
  \[
  \text{fact} \{ \text{all } p: \text{Person} \mid p.\text{father} \neq p \ \text{and} \ \ p.\text{mother} \neq p \}
  \]

• A little awkward, defining each constraint for both father and mother
  • Parent makes more sense
Predicates

• Let’s define a new predicate:

```plaintext
pred parent(p1: Person, p2:Person) {
    p1 = p2.father or
    p1 = p2.mother
}
```
Predicates

• Let’s define a new predicate:

```java
pred parent(p1: Person, p2:Person) {
    p1 = p2.father or
    p1 = p2.mother
}
```

• And use it in some constraints:

```java
fact { all p: Person | !parent[p , p] }
fact { all p1, p2: Person | ! (parent[p1 , p2] and parent[p2, p1] ) }
```
Predicates

• Let’s define a new predicate:

```plaintext
def pred parent(p1: Person, p2:Person) {
    p1 = p2.father or
    p1 = p2.mother
}
```

• And use it in some constraints:

```plaintext
fact { all p: Person | !parent[p , p] }  
fact { all p1, p2: Person | ! (parent[p1 , p2] and
                                 parent[p2, p1] ) } 
```

• Can also get an instance that satisfies the predicate

```plaintext
run parent for 5
```
Functions

• A person shouldn’t be able to be their own grandparent as well
Functions

• A person shouldn’t be able to be their own grandparent as well
• Let’s define a grandparents function:

```haskell
fun grands (p: Person): set Person {
  p.(father+mother).(father+mother)
}
```
Functions

• A person shouldn’t be able to be their own grandparent as well

• Let’s define a grandparents function:

```
fun grands (p: Person): set Person {
  p.(father+mother). (father+mother)
}
```

• And use it in a constraint:

```
fact { no p: Person | p in grands[p] } 
```
Let's check something

- Siblings shouldn't be each other's parents
Let's check something

• Siblings shouldn’t be each others parents
• Let’s define an assertion:

```
assert sibsNotParents {
    no p1, p2: Person |
    some p3: Person |
        parent[p3, p1] and
        parent[p3, p2] and
        parent[p1, p2] and p1 != p2
}
```
Lets check something

• Siblings shouldn’t be each others parents

• Let’s define an assertion:

```plaintext
assert sibsNotParents { 
  no p1, p2: Person |
  some p3: Person |
  parent[p3, p1] and 
  parent[p3, p2] and 
  parent[p1, p2] and p1 != p2
}
```

• And check our assertion:

```plaintext
check sibsNotParents for 5
```
Quantifiers

• **some** – same as $\exists$, one or more elements (match the property)

• **all** – same as $\forall$, all elements (match the property)

• **no** – same as $\neg \exists$, there exist no elements (match the property)

• **one** – exactly one element (matches the property)

• **lone** – zero or one element (match the property)
Transitive Closure

- \( p.\text{mother} \)
  \( p.\text{mother.\text{mother}} \)
  \( p.\text{mother.\text{mother.\text{mother}}} \)
  ...

- \( p.^{\text{mother}} \) – a set of persons, closed under the *mother* operation

- \( p.\text{.*mother} \) – a set of persons, closed under the *mother* operation, includes \( p \)
Transitive Closure

- Let’s get rid of cycles in the family tree
  
  \[
  \text{fact} \{ \text{no } p: \text{Person} \mid p \text{ in } p.^(\text{mother } + \text{ father}) \}\]

- + is a union of relations
- ^ creates a transitive closure
- . performs a join