CIS 771: Software Specifications

Lecture 7: More Dynamic Models in Alloy

Dynamic models in Alloy

- More on dynamic models
  - Operators that generate elements
- Analyzing operations
  - Checking operations against invariants
  - Approaches to frame conditions
- Programs versus specifications
- Mutabilities
  - For defining global frame conditions
- Methodology for operator development
For you to do (pause here)

- Load family-5.all
- Select the marriage operation and run the analyzer
- Does the instance generated include the state changes encoded in the operation?
- Are there any other state changes?
- Are they appropriate?

Instance of Marriage

Person = \{P0, P1, P2\}
Man = \{P0, P2\}
Woman = \{P1\}
Single = \{P0, P1, P2\}
Married = {} 
children = \{P2 -> P0\}
wife = {} 
siblings = {}
**Operation Instantiation**

- Enforces the state and invariant schema for the *before* state
- *After* state is determined by primed sub-formulae in operation
- Is the after state desirable and legal?
  - Operation could be *too weak* : strengthen it
  - Could *violate invariant* : check it
  - We'll look at how to do both in Alloy/ACA
    - You should use version 1.1 of ACA

---

**Specifications versus Programs**

```plaintext
// Next-state formula in a specification
groom.wife' = bride
Married' = Married+groom+bride

// Assignment statements in a program
groom.wife = bride
Married = Married+groom+bride
```

Have a similar effect on the LHS of the equation/assignment, but ...
 Specifications versus Programs

- **Explicit Effects**
  - LHS of assignment (implicitly primed)
  - Primed component in formula

- **Implicit effects**
  - No other effects for assignment
  - Unspecified additional effects for formula

- **Unexpected instance of Marriage**
  
  ```
  children = {P2 -> {P0}}
  children' = {P0 -> {P1}}
  ```

 Specifications versus Programs

- **Deterministic sequential programs**
  - Behavior is determined exactly by the sequence of program statements

- **Non-deterministic Specifications**
  - Choose some value from domain (like an existential quantification)
  - Loose constraints model lack of knowledge
Frame Conditions

Frame conditions define the parts of system state that remain unchanged across an instance of an operation

Marriage Revisited

```op Marriage(groom : Man!, bride : Woman!) {
groom+bride in Single
not BloodRelated(groom, bride)
groom.wife’ = bride
Married’ = Married+bride+groom

// Frame conditions
Person’ = Person
Man’ = Man
all p | p.children’ = p.children
}```
Operation-specific Frame Conditions

```
// Frame conditions
Person' = Person
Man' = Man
\forall p \mid p. children' = p. children
```

- Instantiating an operation includes state and invariant schema
  - it is ok to exploit their constraints
  - e.g., siblings is defined in terms of parent, parent is the inverse of children

---

For you to do (pause here)

- Recall the academia model and the course add operation you were to design
  - What state components change?
  - What components should remain unchanged?
- Design a completed course operation to capture what happens when you are done taking a course
  - What state components change?
  - What state components should remain unchanged?
For you to do (pause here)

- Load family-6.all
- Select the framed marriage operation and run the analyzer
- Does the instance generated include the state changes encoded in the operation?
- Are there any other state changes?

Global Frame Conditions

- Some frame conditions are common to all operations
  - e.g., FixedResources in airport model never change

- We can define domains and sets to be fixed
  
  domain \{fixed FixedResource\}

  partition Plane, Vehicle : fixed FixedResource
Mutabilities

- There are two kinds of mutabilities in Alloy
  - fixed and static
- Mutabilities desugar to core language constraints in all operations/assertions

```alloy
domain {fixed FixedResource}
  FixedResource' = FixedResource
partition Plane, Vehicle : fixed FixedResource
  Plane' = Plane
  Vehicle' = Vehicle
```

Static Sets

- Unlike fixed sets these can change
  - but, how they change is constrained
- For example
  ```alloy
  S : static T
  ```
  - S is a static classification of objects in T
  - objects in T before and after a state transition cannot move into (out of) S
Static Sets

- $S$ : static $T$
  $S = \{p0, p1\}$, $T = \{p0,p1,p2\}$

- Legal transitions?

  $S’ = \{p0, p1\}$, $T’ = \{p0,p1,p2,p3\}$ OK
  $S’ = \{p0, p1,p3\}$, $T’ = \{p0,p1,p2,p3\}$ OK
  $S’ = \{p0, p1,p2\}$, $T’ = \{p0,p1,p2\}$ NOT OK
  $S’ = \{p0\}$, $T’ = \{p0,p1,p2\}$ NOT OK
  $S’ = \{p0\}$, $T’ = \{p0,p2\}$ OK

For you to do (pause here)

- Are there any sets from the academia example that ought to be static?
- What about from the airport example?
Desugaring Static Sets

- Static sets can be expressed in terms of the core language as well
  \[ S : \text{static } T \]

  \[
  \forall e : T \& T' | \\
  e \in S \rightarrow e \in S' \land \\
  e \in S' \rightarrow e \in S
  \]

Mutable Relations

- Mutabilities can be applied to a relation’s image
  \[ \text{rel} : S \rightarrow \text{static } T \]

- **Intuitively**, each unchanging element of \( S \) must map to the same subset of \( T \)

  \[
  \forall s : S \& S' | s.\text{rel} = s.\text{rel}'
  \]
Mutable Relations

- Mutabilities can be applied to a relations domain
  \[ \text{rel} : \text{static } S \rightarrow T \]

- Intuitively, each unchanging element of \(T\) must be mapped to by the same subset of \(S\)

\[ \text{all } t : T \& T' \mid t.\sim \text{rel} = t.\sim \text{rel}' \]

For you to do (pause here)

- Are there any relations from the academia example that ought to be static?
- What if students cannot switch majors?
- Write out the mutabilities for any such relations in the academia model
Birth Transition

- Let’s design a birth transition
- We need
  - Man and Woman who aren’t blood related
- We create
  - A new Person who’s their child
- How about frame conditions?

Birth Frame Conditions

<table>
<thead>
<tr>
<th>Components</th>
<th>Change</th>
<th>Increase</th>
<th>One or other</th>
<th>Add to single</th>
<th>Add child</th>
<th>Add sibling</th>
</tr>
</thead>
<tbody>
<tr>
<td>Person</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Man, Woman</td>
<td>Change</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Single, Married</td>
<td>Change</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>children, parents</td>
<td>Change</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>wife, husband</td>
<td>No Change</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>siblings</td>
<td>Change</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Model-wide Frame Conditions

How will components change for any operation?

<table>
<thead>
<tr>
<th>Component</th>
<th>Change</th>
<th>Unconstrained</th>
</tr>
</thead>
<tbody>
<tr>
<td>Person</td>
<td>Change</td>
<td>Static</td>
</tr>
<tr>
<td>Man, Woman</td>
<td>Change</td>
<td>Static</td>
</tr>
<tr>
<td>Single, Married</td>
<td>Change</td>
<td>Unconstrained</td>
</tr>
<tr>
<td>children, parents</td>
<td>Change</td>
<td>Left static</td>
</tr>
<tr>
<td>wife, husband</td>
<td>Change</td>
<td>Unconstrained</td>
</tr>
<tr>
<td>siblings</td>
<td>Change</td>
<td>Left static</td>
</tr>
</tbody>
</table>

State Modifications for Birth

```
state {
    partition Man, Woman : static Person
    partition Single, Married : Person
    wife (~husband) : Man ? -> Woman ?
    children (~parents) : static Person -> Person
    siblings : static Person -> Person
}
```
**Birth Operation**

\[
\text{op Birth}(\text{dad} : \text{Man!}, \text{mom} : \text{Woman!}, \text{baby} : \text{Person'!}) \{ \\
\quad \text{baby } \uparrow\text{in Person} \\
\quad \text{not BloodRelated(mom,dad)} \\
\quad \text{Person'} = \text{Person} + \text{baby} \\
\quad \text{Single'} = \text{Single} + \text{baby} \\
\quad \text{mom.children'} = \text{mom.children} + \text{baby} \\
\quad \text{dad.children'} = \text{dad.children} + \text{baby} \\
\quad \text{all } p : \text{Person} \mid p.\text{wife'} = p.\text{wife}
\}
\]

**Boy or Girl?**

\[
\text{Person'} = \text{Person} + \text{baby} \\
\text{Single'} = \text{Single} + \text{baby} \\
\text{mom.children'} = \text{mom.children} + \text{baby} \\
\text{dad.children'} = \text{dad.children} + \text{baby}
\]

How is the gender of baby specified?

Non-determinism is useful in defining system interactions with external entities for which no specification exists.
For you to do (pause here)

- Load family-7.all
- Select the birth operation and run the analyzer
- Does the instance generated include the state changes encoded in the operation?
- Are there any other state changes?

Operation Instantiation = Testing

- Instance of birth looks ok
  - Had a girl
  - No unexpected changes
- Are all instances of birth ok?
  - Could force tool to produce an instance
  - Add a condition-like schema to operation
    - e.g., baby in Man'
  - Not a complete check
Operations vs. Invariants

- Current state satisfies state schema
  - And all relevant invariants
- Is there a next state that violates an invariant?
- Counter-examples only include components that are relevant to the violation

For you to do (pause here)

- Load family-7.all
- Select the birth operation and then while pressing the <control> key select the SelfAncestor invariant
- Make sure you are running ver. 1.1 of ACA
- Now run the analyzer
- Is there a counter-example?
  - Does it reveal a missing frame condition?
Counter-example

Analyzing Birth vs. SelfAncestor ...
Scopes: Person(3)
Conversion time: 0 seconds
Solver time: 30 seconds
Counterexample found:
Domains:
Person = {P1,P2}  Person' = {P0,P1,P2}
Sets:
Man = {P2}  Man' = {P0,P2}
Married = {P1,P2}  Married' = {P1,P2}
Single = {}  Single' = {P0}
Woman = {P1}  Woman' = {P1}
Relations:
children = {}  children' = {P0 -> {P0}, P1 -> {P0}, P2 -> {P0}}
husband = {}  husband' = {}
pARENTS = {}  parents' = {P0 -> {P0,P1,P2}}
siblings = {}  siblings' = {P2 -> {P0}}
wife = {}  wife' = {}
Parameters:
baby = P0
dad = P2
mom = P1

Birth Operation

op Birth(dad : Man!, mom : Woman!, baby : Person'!) {
    baby !in Person
    not BloodRelated(mom,dad)
    Person' = Person+baby
    Single' = Single+baby
    mom.children' = mom.children + baby
    dad.children' = dad.children + baby
    no baby.children'
    all p : Person' | p.wife' = p.wife
}
General Methodology

- Make pre and post conditions clear
  - Basis for design contract
- Encode as many frame conditions as possible with mutabilities
- Encode operation specific frame conditions
  - Mutable domains require special care
- Can you make the pre condition true?
  - Instantiate operation
- Does the post condition clash?
  - Check operation against invariant

Specification For Analysis

- When instantiating minimize scope
  - If no instance, then increase scope
  - Smaller instances are easier to read
- When verifying take care with scopes
  - Overly small will yield false positives
  - Incrementally increase scope until you get tired of waiting
- Define individual named invariants
  - All are included with state schema
  - Selective checking against operations is MUCH faster