CS:5810
Formal Methods in Software Engineering

Dynamic Models in Alloy
Overview

• Basics of dynamic models
  – Modeling a system’s states and state transitions
  – Modeling operations causing transitions

• Simple example of operations
Static Models

• So far we’ve used Alloy to define the allowable values of state components
  – values of sets
  – values of relations
• A model instance is a set of state component values that
  – Satisfies the constraints defined by multiplicities, fact, “realism” conditions, ...
Static Models

Person = \{Matt, Sue\}
Man = \{Matt\}
Woman = \{Sue\}
Married = \{Matt, Sue\}
spouse = \{(Matt, Sue), (Sue, Matt)\}
children = {} 
siblings = {}

Person = \{Matt, Sue, Sean\}
Man = \{Matt, Sean\}
Woman = \{Sue\}
Married = \{Matt, Sue\}
spouse = \{(Matt, Sue), (Sue, Matt)\}
children = \{(Matt, Sean), (Sue, Sean)\}
siblings = {}
Dynamic Models

• Static models allow us to describe the legal states of a dynamic system

• We also want to be able to describe the legal transitions between states

  E.g.
  – To get married one must be alive and not currently married
  – One must be alive to be able to die
  – A person becomes someone’s child after birth
Example

*Family Model*

```plaintext
abstract sig Person {
  children: set Person,
  siblings: set Person
}

sig Man, Woman extends Person {}

sig Married in Person {
  spouse: one Married
}
```
State Transitions

• Two people get married

  – At time $t$, $\text{spouse} = \emptyset$
  – At time $t'$, $\text{spouse} = \{(\text{Matt, Sue}), (\text{Sue, Matt})\}$

$\Rightarrow$ We add the notion of time in the relation spouse

<table>
<thead>
<tr>
<th>Time $t$</th>
<th>Time $t'$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Person = {Matt, Sue}</td>
<td>Person = {Matt, Sue}</td>
</tr>
<tr>
<td>Man = {Matt}</td>
<td>Man = {Matt}</td>
</tr>
<tr>
<td>Woman = {Sue}</td>
<td>Woman = {Sue}</td>
</tr>
<tr>
<td>Married = {}</td>
<td>Married = {Matt, Sue}</td>
</tr>
<tr>
<td>spouse = {}</td>
<td>spouse = {(Matt, Sue), (Sue, Matt)}</td>
</tr>
<tr>
<td>children = {}</td>
<td>children = {}</td>
</tr>
<tr>
<td>siblings = {}</td>
<td>siblings = {}</td>
</tr>
</tbody>
</table>
Modeling State Transitions

• Alloy has no predefined notion of state transition.

• However, there are several ways to model dynamic aspects of a system in Alloy.

• A general and relatively simple way is to:
  1. introduce a *Time* signature expressing time.
  2. add a time component to each relation that changes over time.
Family Model Signatures

abstract sig Person {
  children: set Person,
  siblings: set Person set
}
sig Man, Woman extends Person {}

sig Married in Person {
  spouse: one Married one
}
sig Time {}

abstract sig Person {
  children: Person set -> Time,
  siblings: Person set -> Time
}
sig Man, Woman extends Person {}

sig Married in Person {
  spouse: Married one -> Time
}
Transitions

- Two people get married

  - At time $t$, $\text{Married} = \{\}$
  - At time $t'$, $\text{Married} = \{\text{Matt, Sue}\}$

  - Actually, we can’t have a time-dependent signature such as $\text{Married}$ because signatures are not time dependent.

Person = \{\text{Matt, Sue}\}
Man = \{\text{Matt}\}
Woman = \{\text{Sue}\}
Married = \{\}
spouse = \{\}
children = \{\}
siblings = \{\}

Person = \{\text{Matt, Sue}\}
Man = \{\text{Matt}\}
Woman = \{\text{Sue}\}
Married = \{\text{Matt, Sue}\}
spouse = \{(\text{Matt, Sue}), (\text{Sue, Matt})\}
children = \{\}
siblings = \{\}

Time $t$

Time $t'$
Transitions

• A person is born
  – At time $t$, $\text{Person} = \{}$
  – At time $t'$, $\text{Person} = \{\text{Sue}\}$
  – We cannot add the notion being born to the signature $\text{Person}$ because signatures are not time dependent
Signatures are Static

abstract sig Person {
  children: Person set -> Time,
  siblings: Person set -> Time,
  spouse: Person lone -> Time
}
sig Man, Woman extends Person {}
Signatures are Static

abstract sig Person {
    children: Person set -> Time,
    siblings: Person set -> Time,
    spouse: Person lone -> Time
    alive: set Time
}

sig Man, Woman extends Person {}
Revising Constraints

abstract sig Person {
    children: Person set -> Time,
    siblings: Person set -> Time,
    spouse: Person lone -> Time,
    alive: set Time
}

sig Man, Woman extends Person {}

fun parents[] : Person->Person {~children}
Revising Constraints

abstract sig Person {
  children: Person set -> Time,
  siblings: Person set -> Time,
  spouse: Person lone -> Time,
  alive: set Time
  parents: Person set -> Time
}

sig Man, Woman extends Person {}

fun parents[] : Person->Person {~children}

fact parentsDef {
  all t: Time | parents.t = ~(children.t)
}
Revising Constraints

-- Time-dependent parents relation

fact parentsDef {  
  all t: Time | parents.t = ~(children.t) 
}

-- Two persons are blood relatives iff  
-- they have a common ancestor

pred BloodRelatives [p, q: Person, t: Time] {  
  some p.*(parents.t) & q.*(parents.t) 
}
Revising *Static* Constraints

-- People cannot be their own ancestors

\[
\text{all } t: \text{Time} \mid \text{no } p: \text{Person} \mid \\
\text{p in } p.^(\text{parents}.t)
\]

-- No one can have more than one father
-- or mother

\[
\text{all } t: \text{Time} \mid \text{all } p: \text{Person} \mid \\
\text{lone } (p.\text{parents}.t \& \text{Man}) \\
\text{and} \\
\text{lone } (p.\text{parents}.t \& \text{Woman})
\]

...
Revising *Static* Constraints

-- A person p's siblings are those people, other than p, with the same parents as p

\[
\text{all } t: \text{Time} \mid \text{all } p: \text{Person} \mid \\
p.\text{siblings}.t = \\
\{ \text{q: Person} - p \mid \text{some q.parents}.t \text{ and} \\\n\quad p.\text{parents}.t = q.\text{parents}.t \} \\
\]

-- Each married man (woman) has a wife (husband)

\[
\text{all } t: \text{Time} \mid \text{all } p: \text{Person} \mid \\
\text{let } s = p.\text{spouse}.t \mid \\
(p \text{ in Man implies } s \text{ in Woman}) \text{ and} \\
(p \text{ in Woman implies } s \text{ in Man})
\]
Revising *Static* Constraints

-- A spouse can't be a sibling

\[
\text{all } t: \text{Time} \mid \text{no } p: \text{Person} \mid \\
\text{some } p.\text{spouse.t and} \\
p.\text{spouse.t in p.siblings.t}
\]

-- People can't be married to a blood relative

\[
\text{all } t: \text{Time} \mid \text{no } p: \text{Person} \mid \\
\text{let } s = p.\text{spouse.t} \mid \\
\text{some } s \text{ and} \\
\text{BloodRelatives}[p, s, t]
\]
Revising Static Constraints

-- a person can't have children with
-- a blood relative
\[
\text{all } t: \text{Time} \mid \text{all } p, q: \text{Person} \mid
(\text{some } (p.\text{children}.t \& q.\text{children}.t) \text{ and } p \neq q)
\implies
\text{not } \text{BloodRelatives}[p, q, t]
\]

-- the spouse relation is symmetric
\[
\text{all } t: \text{Time} \mid
\text{spouse}.t = \sim(\text{spouse}.t)
\]
Exercises

• Load family-6.als
• Execute it
• Analyze the model
• Look at the generated instance
• Does it look correct?
• What, if anything, would you change about it?
Transitions

A person is born from parents

– Add to alive relation

– Modify children/parents relations

Person = \{Matt, Sue, Sean\}
Man = \{Matt, Sean\}
Woman = \{Sue\}
spouse = \{(Matt,Sue), (Sue,Matt)\}
children = \{\}
siblings = \{\}
alive = \{Matt, Sue\}

Person = \{Matt, Sue, Sean\}
Man = \{Matt, Sean\}
Woman = \{Sue\}
spouse = \{(Matt,Sue), (Sue,Matt)\}
children = \{(Matt,Sean), (Sue,Sean)\}
siblings = \{\}
alive = \{Matt, Sue, Sean\}
State Sequences

Person = \{Matt, Sue, Sean\}
Man = \{Matt, Sean\}
Woman = \{Sue\}
spouse = {} 
children = {} 
siblings = {} 
alive = \{Sue\}
Expressing Transitions in Alloy

• A transition can be thought of as caused by the application of an operator to the current state

• An operator can be modeled as predicates between two states:
  1. the state right before the transition and
  2. the state right after it

• We define it as predicate with (at least) two formal parameters: \( t, t' : \text{Time} \)

• Constraints over time \( t \) (resp., \( t' \)) model the state right before (resp., after) the transition
Expressing Transitions in Alloy

• **Pre-condition constraints**
  – Describe the states to which the transition applies

• **Post-condition constraints**
  – Describes the effects of the transition in generating the next state

• **Frame-condition constraints**
  – Describes what does not change between pre-state and post-state of a transition

*Distinguishing the pre-, post- and frame-conditions in comments provides useful documentation*
Example: Marriage

```plaintext
pred getMarried [m: Man, w: Woman, t,t': Time] {  
  -- preconditions  
  -- m and w must be alive  
  m+w in alive.t  
  -- neither one is married  
  no (m+w).spouse.t  
  -- they are not be blood relatives  
  not BloodRelatives[m, w, t]  
  -- post-conditions  
  -- w is m’s wife  
  m.spouse.t' = w  
  -- m is w’s husband  
  w.spouse.t' = m  
  -- frame conditions ??  
}
```
How is each relation touched by marriage?

- 5 relations:
  - children, parents, siblings
  - spouse
  - alive

- parents and siblings relations are defined in terms of the children relation

- Thus, the frame condition has only to consider children, spouse and alive relations
Frame Condition Predicates

\texttt{pred noChildrenChangeExcept } [ps: set Person t,t': Time] \{ \\
all p: Person - ps | \\
\quad p\.children\.t' = p\.children\.t \\
\}

\texttt{pred noSpouseChangeExcept } [ps: set Person t,t': Time] \{ \\
all p: Person - ps | \\
\quad p\.spouse\.t' = p\.spouse\.t \\
\}

\texttt{pred noAliveChange } [t,t': Time] \{ \\
\quad alive\.t' = alive\.t \\
\}
Example: Marriage

pred getMarried[m: Man, w: Woman, t,t': Time] {
  -- preconditions
  m+w in alive.t
  no (m+w).spouse.t
  not BloodRelatives[m, w, t]
  -- post-conditions
  m.spouse.t' = w
  -- frame conditions
  noChildrenChangeExcept[none, t, t’]
  noSpouseChangeExcept[m+w, t, t’]
  noAliveChange[t, t’]
}
Instance of Marriage

\[
\text{open ordering [Time] as T}
\]

\[
\text{...}
\]

\[
\text{pred marriageInstance}\{
\text{some t: Time |}
\text{some m: Man | some w: Woman |}
\text{getMarried[m, w, t, T/next[t]]}
\}
\]

\[
\text{run}\{\text{marriageInstance}\}
\]
Example: Birth from Parents

```
pred isBornFromParents [p: Person, m,w: Person, t,t': Time] {
    -- Pre-condition
    m+w in alive.t
    p !in alive.t

    -- Post-condition and frame condition
    alive.t' = alive.t + p
    m.children.t' = m.children.t + p
    w.children.t' = w.children.t + p

    -- Frame condition
    noChildrenChangeExcept[m+w, t, t']
    noSpouseChangeExcept[none, t, t']
}
```
Instance of Birth

```plaintext
def birthInstance {  
some t: Time  |
  some p1, p2, p3: Person  |
    isBornFromParents[p1, p2, p3, t, T/next[t]]
}

def run { birthInstance }
```
Example: Death

\texttt{pred} \texttt{dies [p: Person, t, t\': Time] \{ \\
  -- Pre-condition \\
  p \texttt{in alive.t} \\
  \\
  -- Post-condition \\
  \texttt{no p.spouse.t'} \\
  \\
  -- Post-condition and frame condition \\
  \texttt{alive.t'} = \texttt{alive.t} - p \\
  \texttt{all s: p.spouse.t |} \\
  \texttt{s.spouse.t'} = \texttt{s.spouse.t} - p \\
  \\
  -- Frame condition \\
  \texttt{noChildrenChangeExcept[none, t, t']} \\
  \texttt{noSpouseChangeExcept[p + p.spouse.t, t, t']} \\
\}}
Instance of Death

pred deathInstance {
    some t: Time |
    some p: Person |
    dies[p, t, T/next[t]]
}

run { deathInstance }
Specifying Transition Systems

• A transition system can be defined as a set of executions:
  sequences of time steps generated by the operators

• In our example, for every execution:
  – The first time step satisfies some initialization condition
  – Each pair of consecutive steps are related by
    • a birth operation, or
    • a death operation, or
    • a marriage operation
Initial State Specification

`init` specifies constraints on the initial state

```plaintext
pred init [t: Time] {
  no children.t
  no spouse.t
  #alive.t > 2
  #Person > #alive.t
}
```
Transition Relation Specification

`trans` specifies that each transition is a consequence of the application of one of the operators to some individuals.

```plaintext
pred trans [t,t': Time] { 
  (some m: Man, w: Woman | 
    getMarried [m, w, t, t'])
  or 
  (some p: Person, m: Man, w: Woman | 
    isBornFromParents [p, m, w, t, t'])
  or 
  (some p :Person | dies [p, t, t'])
}
```
System Specification

System specifies that each execution of the system start in a state satisfying the initial state condition and move from one state to the next through one transition at a time, until it reaches the final state.

pred System {
    init[T/first]
    all t: Time - T/last | trans[t, T/next[t]]
}

run { System }
System Invariants

• Many of the facts that we stated in our static model now become expected system invariants

• These are properties that
  – should hold in initial states
  – should be preserved by system transitions

• In Alloy we can check that a property is invariant (in a given scope) by
  – encoding it as a formula $P$ and checking
  – checking the assertion

  $\text{System} \Rightarrow \forall t: \text{Time} \mid P$
Expected Invariants: Examples

-- People cannot be their own ancestors
assert a1 { System => all t: Time |
    no p: Person | p in p.^(parents.t) }
check a1 for 8

-- No one can have more than one father or mother
assert a2 { System => all t: Time |
    all p: Person |
    lone (p.parents.t & Man) and
    lone (p.parents.t & Woman) }
check a2 for 8
Exercises

• Load family-7.als
• Execute it
• Look at the generated instance
• Does it look correct?
• What if anything would you change about it?
• Check each of the given assertions
• Are they all valid?
• If not, how would you change the model to fix that?