Alloy Analyzer 4 Tutorial

Session 2: Language and Analysis

Greg Dennis and Rob Seater
Software Design Group, MIT
alloy language & analysis

• language = syntax for structuring specifications in logic
  – shorthands, puns, sugar

• analysis = tool for finding solutions to logical formulas
  – searches for and visualizes counterexamples
“I'm My Own Grandpa” Song

- popular radio skit originally written in the 1930's
- expanded into hit song by “Lonzo and Oscar” in 1948
module grandpa

abstract sig Person {
    father: lone Man,
    mother: lone Woman
}

sig Man extends Person {
    wife: lone Woman
}

sig Woman extends Person {
    husband: lone Man
}

fact {
    no p: Person |
    p in p.(mother + father)
    wife = ~husband
}

assert noSelfFather {
    no m: Man | m = m.father
}

check noSelfFather

fun grandpas[p: Person] : set Person {
    p.(mother + father).father
}

pred ownGrandpa[p: Person] {
    p in grandpas[p]
}

run ownGrandpa for 4 Person

“I'm My Own Grandpa” in Alloy
language: module header

module grandpa

- first non-comment of an Alloy model
<table>
<thead>
<tr>
<th>sig A</th>
<th>}</th>
</tr>
</thead>
<tbody>
<tr>
<td>set of atoms A</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sig A</th>
<th>}</th>
</tr>
</thead>
<tbody>
<tr>
<td>sig B</td>
<td>}</td>
</tr>
<tr>
<td>disjoint sets A and B (no A &amp; B)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sig A, B</th>
<th>}</th>
</tr>
</thead>
<tbody>
<tr>
<td>same as above</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sig B extends A</th>
<th>}</th>
</tr>
</thead>
<tbody>
<tr>
<td>set B is a subset of A (B in A)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sig B extends A</th>
<th>}</th>
</tr>
</thead>
<tbody>
<tr>
<td>sig C extends A</td>
<td>}</td>
</tr>
<tr>
<td>B and C are disjoint subsets of A</td>
<td></td>
</tr>
<tr>
<td>(B in A &amp; C in A &amp; no B &amp; C)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sig B, C extends A</th>
<th>}</th>
</tr>
</thead>
<tbody>
<tr>
<td>same as above</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>abstract sig A</th>
<th>}</th>
</tr>
</thead>
<tbody>
<tr>
<td>sig B extends A</td>
<td>}</td>
</tr>
<tr>
<td>sig C extends A</td>
<td>}</td>
</tr>
<tr>
<td>A partitioned by disjoint subsets B and C</td>
<td></td>
</tr>
<tr>
<td>(no B &amp; C &amp; A = (B + C))</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sig B in A</th>
<th>}</th>
</tr>
</thead>
<tbody>
<tr>
<td>B is a subset of A – not necessarily disjoint from any other set</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sig C in A + B</th>
<th>}</th>
</tr>
</thead>
<tbody>
<tr>
<td>C is a subset of the union of A and B</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>one sig A</th>
<th>}</th>
</tr>
</thead>
<tbody>
<tr>
<td>lone sig B</td>
<td>}</td>
</tr>
<tr>
<td>some sig C</td>
<td>}</td>
</tr>
<tr>
<td>A is a singleton set</td>
<td></td>
</tr>
<tr>
<td>B is a singleton or empty</td>
<td></td>
</tr>
<tr>
<td>C is a non-empty set</td>
<td></td>
</tr>
</tbody>
</table>
all men and women are persons
no person is both a man and a woman
all persons are either men or women
**language: fields**

\[
\text{sig } A \set \{ f : e \}
\]

\( f \) is a binary relation with domain \( A \) and range given by expression \( e \)

\( f \) is constrained to be a function

\( (f : A \rightarrow \text{one } e) \) or \( (\text{all } a : A | a.f : e) \)

\[
\text{sig } A \set
\begin{align*}
  & f1 : \text{one } e1, \\
  & f2 : \text{lone } e2, \\
  & f3 : \text{some } e3, \\
  & f4 : \text{set } e4
\end{align*}
\]

\( (\text{all } a : A | a.\text{fn} : m e) \)

\[
\text{sig } A \set \{ f, g : e \}
\]

two fields with same constraints

\[
\text{sig } A \set \{ f : e1 m \rightarrow n e2 \}
\]

\( (f : A \rightarrow (e1 m \rightarrow n e2)) \) or

\( (\text{all } a : A | a.f : e1 m \rightarrow n e2) \)

\[
\text{sig Book } \set
\begin{align*}
  & \text{names} : \text{set } \text{Name}, \\
  & \text{addrs} : \text{names } \rightarrow \text{Addr}
\end{align*}
\]

dependent fields

\( (\text{all } b : \text{Book} | b.\text{addrs} : b.\text{names } \rightarrow \text{Addr}) \)
grandpa: fields

abstract sig Person {
    father: lone Man,
    mother: lone Woman
}

sig Man extends Person {
    wife: lone Woman
}

sig Woman extends Person {
    husband: lone Man
}

- fathers are men and everyone has at most one
- mothers are women and everyone has at most one
- wives are women and every man has at most one
- husbands are men and every woman has at most one
language: facts

```
fact { F }
fact f { F }
sig S { ... }{ F }
```

facts introduce constraints that are assumed to always hold

```
sig Host {}
sig Link {from, to: Host}

fact {all x: Link | x.from != x.to}
no links from a host to itself

fact noSelfLinks {all x: Link | x.from != x.to}
same as above

sig Link {from, to: Host} {from != to}
same as above, with implicit 'this.'
```
grandpa: fact

- no person is his or her own ancestor
- a man's wife has that man as a husband
- a woman's husband has that woman as a wife
**language: functions**


```
fun f[x1: e1, ... , xn: en] : e { E }
```

*functions are named expression with declaration parameters and a declaration expression as a result invoked by providing an expression for each parameter*

```
sig Name, Addr {}
sig Book {   
  addr: Name -> Addr
}

fun lookup[b: Book, n: Name] : set Addr {   
  b.addr[n]
}

fact everyNameMapped {   
  all b: Book, n: Name | some lookup[b, n]
}
```
language: predicates

\[ \textbf{pred} \ p[x_1: e_1, \ldots, x_n: e_n] \{ F \} \]

\textbf{named formula with declaration parameters}

\begin{verbatim}
sig Name, Addr {}
sig Book {
    addr: Name -> Addr
}

pred contains[b: Book, n: Name, d: Addr] {
    n->d \in b.addr
}

fact everyNameMapped {
    all b: Book, n: Name |
    some d: Addr | contains[b, n, a]
}
\end{verbatim}
grandpa: function and predicate

fun grandpas[p: Person] : set Person {
  p.(mother + father).father
}

pred ownGrandpa[p: Person] {
  p in grandpas[p]
}

• A person's grandpas are the fathers of one's own mother and father
language: “receiver” syntax

\[
\begin{align*}
\text{fun } & f[x: X, y: Y, \ldots] : Z \{\ldots x \ldots\} \\
\text{fun } & X.f[y:Y, \ldots] : Z \{\ldots \text{this} \ldots\} \\
\text{pred } & p[x: X, y: Y, \ldots] \{\ldots x \ldots\} \\
\text{pred } & X.p[y:Y, \ldots] \{\ldots \text{this} \ldots\}
\end{align*}
\]

\[
\begin{align*}
\text{fun } & \text{Person.grandpas} : \text{set} \text{ Person} \{ \\
& \quad \text{this.(mother + father).father} \\
& \} \\
\text{pred } & \text{Person.ownGrandpa} \{ \\\n& \quad \text{this in this.grandpas} \\
& \}
\end{align*}
\]
assert a { F }

class Node {
    children: set Node
}

one class Root extends Node {}

@fact {
    Node in Root.*children
}

// invalid assertion:
assert someParent {
    all n: Node | some children.n
}

// valid assertion:
assert someParent {
    all n: Node - Root | some children.n
}
language: check command

assert a { F }
check a scope

if model has facts M
finds solution to M && !F

check a
top-level sigs bound by 3

check a for default
top-level sigs bound by default

check a for default but list
default overridden by bounds in list

check a for list
sigs bound in list,
invalid if any unbound

abstract sig Person {}
sig Man extends Person {}
sig Woman extends Person {}
sig Grandpa extends Man {}

check a
check a for 4
ccheck a for 4 but 3 Woman
ccheck a for 4 but 3 Man, 5 Woman
ccheck a for 4 Person
ccheck a for 4 Person, 3 Woman
ccheck a for 3 Man, 4 Woman
ccheck a for 3 Man, 4 Woman, 2 Grandpa

// invalid:
ccheck a for 3 Man
ccheck a for 5 Woman, 2 Grandpa
grandpa: assertion check

\[
\text{fact} \{ \\
\qquad \text{no } p : \text{Person} \mid p \text{ in } p.^(\text{mother + father}) \\
\qquad \quad \text{wife} = \sim\text{husband} \\
\} \\
\]

\[
\text{assert } \text{noSelfFather} \{ \\
\qquad \text{no } m : \text{Man} \mid m = m.\text{father} \\
\} \\
\]

\[
\text{check } \text{noSelfFather} \\
\]

- sanity check
- command instructs analyzer to search for counterexample to \textit{noSelfFather} within a scope of at most 3 \textit{Persons}
- \textit{noSelfFather} assertion follows from fact
language: run command

**pred** \( p[x: X, y: Y, ...] \) \{ \( F \) \}
**run** \( p \) **scope**

*instructs analyzer to search for instance of predicate within scope*

if model has facts \( M \), finds solution to \( M \) && (some \( x: X, y: Y, ... \) \| \( F \) )

**fun** \( f[x: X, y: Y, ...] : R \) \{ \( E \) \}
**run** \( f \) **scope**

*instructs analyzer to search for instance of function within scope*

if model has facts \( M \), finds solution to \( M \) && (some \( x: X, y: Y, ..., result: R \| result = E \))
grandpa: predicate simulation

fun grandpas[p: Person] : set Person {
    p.(mother + father).father
}

pred ownGrandpa[p: Person] {
    p in grandpas[p]
}

run ownGrandpa for 4 Person

- command instructs analyzer to search for configuration with at most 4 people in which a man is his own grandfather
exercise: barber paradox

- download *barber.als* from the tutorial website
- follow the instructions
- don't hesitate to ask questions

```plaintext
sig Man {shaves: set Man}
one sig Barber extends Man {}
fact {
    Barber.shaves = {m: Man | m not in m.shaves}
}
```
introduction to visualization

➢ Download *grandpa.als* from the tutorial website
➢ Click “Execute”
➢ Click “Show”
➢ Click “Theme”
superficial

- types and sets
  - default color $\rightarrow$ gray
  - Apply
  - man color $\rightarrow$ blue
  - woman color $\rightarrow$ red
  - Apply

- also notice:
  - hide unconnected nodes
  - orientation
  - layout backwards
types & sets

- types: from signatures
  - person shape → trapezoid
  - notice it carries down to man, woman
  - woman: align by type
  - *Apply*
types & sets
types & sets

- sets: from existentials, runs, checks
  - somewhat intelligently named
  - $ownGrandpa_m$ label $\rightarrow$ self-grandpa
  - *Apply*

- pitfall: don't show vs. don't show as label
  (vs. don't show in customizer...)

\[ 
\begin{array}{c}
\text{Man}_0 \\
(own\text{Grandpa}_m) \\
\end{array} \quad \begin{array}{c}
\text{Man}_0 \\
(self\text{-grandpa}) \\
\end{array} 
\]
relations

- relations
  - mother: show as attribute → check
    (still shown as arc)
  - gray = inherited (vs. overridden)
  - Apply
relations

- relations
  - mother: show as attribute $\rightarrow$ uncheck
  - father, mother, husband, wife: label $\rightarrow$ “ ”
  - father, mother: color $\rightarrow$ green
  - husband, wife: color $\rightarrow$ yellow
  - Apply
finishing up

- save theme
- close theme

- create your own visualization for the barber exercise!