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
"I'm My Own Grandpa" in Alloy

```alloy
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).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
```
language: module header

```
module grandpa
```

- first non-comment of an Alloy model
**language: signatures**

\[
\text{\texttt{sig A \{\}}}
\]

*set of atoms A*

\[
\text{\texttt{sig A \{\}}}
\quad \text{\texttt{sig B \{\}}}
\]

*disjoint sets A and B (no A \& B)*

\[
\text{\texttt{sig A, B \{\}}}
\]

*same as above*

\[
\text{\texttt{sig B extends A \{\}}}
\]

*set B is a subset of A (B in A)*

\[
\text{\texttt{sig B extends A \{\}}}
\quad \text{\texttt{sig C extends A \{\}}}
\]

*B and C are disjoint subsets of A
(B in A \&\& C in A \&\& no B \& C)*

\[
\text{\texttt{sig B, C extends A \{\}}}
\]

*same as above*

\[
\text{\texttt{abstract sig A \{\}}}
\quad \text{\texttt{sig B extends A \{\}}}
\quad \text{\texttt{sig C extends A \{\}}}
\]

*A partitioned by disjoint subsets B and C
(no B \& C \&\& A = (B + C))*

\[
\text{\texttt{sig B in A \{\}}}
\]

*B is a subset of A – not necessarily disjoint from any other set*

\[
\text{\texttt{sig C in A + B \{\}}}
\]

*C is a subset of the union of A and B*

\[
\text{\texttt{one sig A \{\}}}
\quad \text{\texttt{lone sig B \{\}}}
\quad \text{\texttt{some sig C \{\}}}
\]

*A is a singleton set
B is a singleton or empty
C is a non-empty set*
grandpa: signatures

abstract sig Person {
    · · ·
}

sig Man extends Person {
    · · ·
}

sig Woman extends Person {
    · · ·
}

• all men and women are persons
• no person is both a man and a woman
• all persons are either men or women
language: fields

\[
\text{sig A \{f: e\}} \quad \text{\textit{f} is a binary relation with domain \textit{A} and range given by expression \textit{e}} \\
\text{\quad \textit{f} is constrained to be a function} \\
\text{\quad (\textit{f}: \textit{A} -> \textit{one} \textit{e}) or (all \textit{a}: \textit{A} \land \textit{a.f}: \textit{e})}
\]

\[
\text{sig A \{f: e\}} \quad \text{\textit{two fields with same constraints}} \\
\text{\quad (\textit{f}: \textit{A} -> (\textit{e1 m} -> \textit{n e2})) or} \\
\text{\quad (all \textit{a}: \textit{A} \land \textit{a.f}: \textit{e1 m} -> \textit{n e2})}
\]

\[
\text{sig Book \{}} \\
\text{\quad \textit{names: set Name,}} \\
\text{\quad \textit{addrs: names -> Addr}} \\
\text{\quad \textit{dependent fields}} \\
\text{\quad (all \textit{b}: Book \land \textit{b.addrs}: \textit{b.names} -> Addr)}
\]
grandpa: fields

```java
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

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

facts introduce constraints that are assumed to always hold

```plaintext
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

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

- 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

```plaintext
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

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

named formula with declaration parameters

\[
sig \text{Name, Addr } \{ \}
sig \text{Book } \{
    \text{addr: Name } \rightarrow \text{Addr }
\}
\]

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

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

```plaintext
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

fun f[x: X, y: Y, ...] : Z {...x...}
fun X.f[y:Y, ...] : Z {...this...}

pred p[x: X, y: Y, ...] {...x...}
pred X.p[y:Y, ...] {...this...}

fun Person.grandpas : set Person {
  this.(mother + father).father
}
pred Person.ownGrandpa {
  this in this.grandpas
}
language: assertions

assert a { F }

constraint intended to follow from facts of the model

```plaintext
sig Node {
    children: set Node
}

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

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

```plaintext
fact {
    no p: Person | p in p.^{mother + father}
    wife = ~husband
}

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

check noSelfFather
```

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

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

--

instructs analyzer to search for instance of predicate within scope

if model has facts \( M \), finds solution to \( M \land \) (some \( x: X, y: Y, \ldots \mid F \))

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

--

instructs analyzer to search for instance of function within scope

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

```plaintext
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
  - \textit{Apply}
  - \textit{man} color $\rightarrow$ blue
  - woman color $\rightarrow$ red
  - \textit{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

Diagram:

- Woman0
- Woman1
- Man1
- Man0 (ownGrandpa_m)

Relationships:
- Woman0 -> Woman1: mother
- Woman1 -> Man1: husband
- Man1 -> Woman0: wife
- Man1 -> Man0: father
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...)
relations

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

• relations
  – mother: show as attribute → uncheck
  – father, mother, husband, wife: label → “ ”
  – father, mother: color → green
  – husband, wife: color → yellow
  – Apply
relations

Woman0

Woman1

Man0
(self-grandpa)

Man1
finishing up

• save theme
• close theme

- create your own visualization for the barber exercise!