Alloy Language & Analysis

Excerpted (mostly), adapted (a bit) from:
SAIL Tutorial at
http://alloy.mit.edu/alloy/tutorials/day-course/
by Greg Dennis and Rob Seater
Alloy = logic + language + analysis

• **logic**
  – first order logic + relational calculus

• **language**
  – syntax for structuring specifications in the logic

• **analysis**
  – bounded exhaustive search for counterexample to a claimed property using SAT
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
    no wife &
    *(mother+father).mother
    no husband &
    *(mother+father).father
}

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

check noSelfFather

fun grandpas [p: Person] : set Person {
    let parent = mother + father +
                father.wife +
                mother.husband |
    p.parent.parent & Man
}

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

**abstract sig** A {}

**sig B extends** A {}
**sig C extends** A {}

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

**sig B in** A {}

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

**sig C in** A + B {}

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

**one sig** A {}
**lone sig** B {}
**some sig** C {}

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

**sig** A {}

* set of atoms A

**sig** A {}

**sig** B {}

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

**sig** A, B {}

* same as above

**sig** B extends A {}

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

**sig** B extends A {}
**sig** C extends A {}

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

**sig** B, C extends A {}

* same as above
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
**sig** A \{f: e\}

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

- \(f\) is constrained to be a function

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

**sig** A \{f, g: m e\}

two fields with same constraints

**sig** A \{f: e1 m \to n e2\}

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

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

**sig** Book \{

  names: set Name,
  addrs: names \to Addr

}\d

dependent fields

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

```alloy
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 }

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

**fact Biology**  
\[
\text{no } \ p: \text{Person} \mid p \in p.^(\text{mother} + \text{father})
\]

**fact Terminology**  
\[
\text{wife} = \sim\text{husband}
\]

**fact SocialConvention**  
\[
\text{no wife} \& \ast (\text{mother+father}).\text{mother}
\]
\[
\text{no husband} \& \ast (\text{mother + father}).\text{father}
\]

- no person is his or her own ancestor
- a man’s [woman’s] wife [husband] has that man [woman] as a husband [wife]
- a man [woman] does not marry one of his [her] ancestors who is also a mother [father]
language: functions

fun \( f[x_1 : e_1, \ldots, x_n : e_n] : e \{ E \} \)

- a function is a 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

pred p[x_1: e_1, ..., x_n: e_n] { F }

named formula with declaration parameters

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]
}
grandpa: function and predicate

fun grandpas[p: Person] : set Person {
  let parent = mother + father +
    father.wife + mother.husband |
  p.parent.parent & Man
}

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

- a one's grandpas are the male parents of one’s own parents,
  where one’s parents are one’s mother, one’s father and one’s
  father’s wife and one’s mother’s husband, if any
assert a { F }  

constraint intended to follow from facts of the model

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**

```plaintext
assert a { F }
check a scope

instructs analyzer to search for
counterexample to assertion within 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 top level sig 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

```alloy
fact Biology {
    no p: Person | p in p.^-(mother + father)
}

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

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

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

**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 
\( M \land (\text{some } x: X, y: Y, ..., \text{result: } R | \text{result} = E) \)
grandpa: predicate simulation

fun grandpas [p: Person] : set Person {
  let parent = mother + father + 
    father.wife + 
    mother.husband | 
    p.parent.parent & Man 
}

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
introduction to visualization

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

- types and sets
  - default color → gray
  - *Apply*
  - *man* color → blue
  - *woman* color → 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
  - $own\text{Grandpa}_m$ 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 → check (still shown as arc)
  - gray = inherited (vs. overridden)
  - Apply
relations

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

- save theme
- close theme

- create your own visualization for the barber exercise!