alloy language & analysis

- language = syntax for structuring specifications in logic
  - shorthands, sugar

- analysis = tool for finding solutions to logical formulas
  - searches for and visualizes counterexamples
module examples/tutorial/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
Sig A {}
set of atoms A

Sig A {}
Sig B {}
disjoint sets A and B (no A & B)

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)

Abstract Sig A {}
Sig B extends A {}
Sig C extends A {}
A 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

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

**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

(all $a: A \mid a.f: e$)

**sig** $A \{ f: e \}$

two fields with same constraints

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

map each member of $e1$ to $n$ members of $e2$

and map $m$ members of $e1$ to each members of $e2$

(all $a: A \mid a.f: e1 \ xrightarrow{m} n \ e2$)

**sig** Book $\{$

names: set Name,
addrs: names $\to$ Addr

$\}$

dependent fields

(all $b: Book \mid b.addrs: b.names \to Addr$)
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

facts introduce constraints that are assumed to always hold

```
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
```
grandpa: fact

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

- 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
functions are named expression with declaration parameters and a declaration expression as a result invoked by providing an expression for each parameter
language: predicates

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

\[
\text{sig Name, Addr } \{}
\]

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

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

\[
\text{fact } \text{everyNameMapped} \{ \text{ all b: Book, n: Name | some d: Addr | contains(b, n, a)} \}
\]
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
assert a { F }

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
```

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
```

```
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
```
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
```

- 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

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

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

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

\textit{if model has facts } M \textit{, finds solution to } M \ &\& (\text{some } x: X, y: Y, ..., \text{ result: R }| \text{ result } = E) \textit{ }

\textit{instructs analyzer to search for instance of predicate within scope}

\textit{instructs analyzer to search for instance of function within scope}
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 Person in which a man is his own grandfather
introduction to visualization

- open examples/tutorial/grandpa.als
- build-execute
- select “layout”
visualization layout pane

- the tabs
  - palette = set of views (lightweight)
  - general (default for inheritance)
  - univ (ignore)
  - modules (for us, just “grandpa”)

![Diagram of visualization layout pane with various settings and options.]
superficial

- general tab
  - default type color → gray
  - update

- also notice:
  - hide unconnected nodes
  - orientation
  - layout backwards
another view . . .

• views tab
  – save palette
  – new view “colored view”
  – automatically selected (but not applied)

• general tab
  – node color → martha

• grandpa tab
  – man color → blue
  – woman color → red
  – update
another view . . .
types & sets

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

- sets: from existentials, runs, checks
  - somewhat intelligently named
  - `ownGrandpa_m` label → self-grandpa
  - update

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

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

- mother: show as attribute → uncheck
- father, mother, husband, wife: label → “”
- father, mother: color → green
- husband, wife: color → yellow
- update
relations
defined variables

- Define var
  - spouse = husband + wife
  - spouse: label = “”, color = yellow
  - husband, wife: show as arcs: uncheck
defined variables

- Define var
  - married = ?
defined variables

- Define var
  - married = Person.(husband + wife)

- update

- handy trick: define in order to hide
finishing up

• views
  – save palette
• close layout

▶ create your own visualization for the barber exercise!