MFES - Métodos Formais em Engenharia de Software

Alloy

Ana Paiva
apaiva@fe.up.pt www.fe.up.pt/~apaiva

Signatures

```
sig A {
  R: B        é o mesmo que ter R: one B
}
```

```
sig A {
  R: B -> C     é o mesmo que ter R: B set -> set C
}
```


enum

- In older versions of Alloy Analyzer, to express an enumeration, like
  enum X { A, B, C }
- you have to define
  abstract sig X { }
  one sig A, B, C extends X { }

Fields

- Relations can be declared as fields
- By default binary relations are functions
- The range can be constrained with a multiplicity

```
abstract sig Object {
        name: Name,
        parent: lone Dir
}
sig File extends Object {}
sig Dir extends Object {}
sig Name {}
```

Fields

- Multirelations can also be declared as fields
- Fields can depend on other fields
- Overloading is allowed for non-overlapping signatures

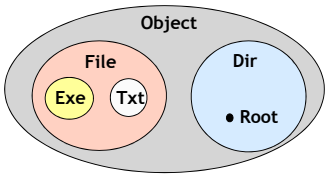
```
abstract sig Object {}
sig File, Dir extends Object {}
sig Name {}
sig FileSystem {
        objects: set Object,
        parent: objects -> lone (Dir & objects),
        name: objects lone -> one Name
}
```


Signatures

- A signature can extend another signature
- The extensions are mutually disjoint
- Signatures can be constrained with a multiplicity

```
sig  Object {}
sig File extends Object {}
sig  Dir extends Object {}
sig Exe, Txt extends File {}
one sig Root extends Dir {}
```

sig X {}
sig A,B extends X{}
$A \cup B \subseteq X$
$A \cap B = \{\}$

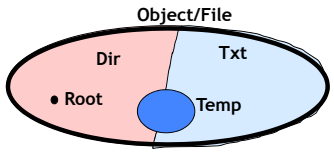


Signatures

- A signature can be abstract
- Arbitrary subset relations can also be declared

```
abstract sig Object {}
abstract sig File extends Object {}
sig Dir, Txt extends File {}
one sig Root extends Dir {}
sig Temp in Object {}
```

abstract sig X {}
sig A,B extends X {}
$A \cap B = \{\}$
$A \cup B = X$



Signatures

abstract sig Person { . . . }

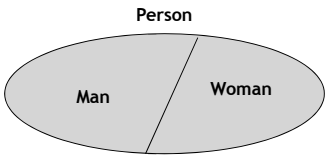
sig Man extends Person { . . .}

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



Signatures

```
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

```
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

```
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

Fact

```
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*

Function/Predicate

```
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

Self-grandpas: an example

…

- Oh, if my wife's my grandmother then I am her grandchild.
- And every time I think of it, it nearly drives me wild.
- For now I have become the strangest case you ever saw
- As the husband of my grandmother, **I am my own grandpa**.

"Self-grandpas"

```
// person has one mother and one father
abstract sig Person {
   father: lone Man,
   mother : lone Woman
}

// one man can be married to one woman
sig Man extends Person {
   wife : lone Woman
}

// one woman can be married to one man
sig Woman extends Person {
   husband : lone Man
}
```


"Self-grandpas"

```
// one person cannot be mother/father of itself
fact Biologia {
   no x : Person | x in x.^(father+mother)
}
// if one man is married to one woman, that woman is married to that man
fact Married {
   wife = ~husband
   no (wife+husband) & ^(father+mother)
}
//there is no person which is gandfather of itself
assert Grandfather {
   no x : Person | x in
   x.(father+mother+mother.father+father.wife).(father+mother.husband)
}
check Avo for 4
run {} for 4
```


"Self-grandpas"