Contents

1 Alloy Reference

1.1 Signatures

A signature defines a set of atoms. Inheritance via `extends` corresponds to a subset relation. `abstract` same as usual. Signatures can have multiplicies.

```
abstract sig FSObject {}

sig File extends FSObject {}
sig Dir extends FSObject {}

one sig Root extends Dir {}
```

1.1.1 Fields

Signatures can contain fields with a multiplicity, which are equivalent to binary relations between the signature and the element type.

```
abstract sig FSObject {
  parent: lone Dir
}

sig Dir extends FSObject {
```

```
  contents: set FSObject
}

sig University {
  students: set Student,
  enrollment: students set -> one Program // Can depend on other field
}
```

1.2 Operations

1.2.1 On sets

- `+` (union)
- `&` (intersection)
- `-` (difference)
- `in` (subset)
- `=` (equality)
- `#` (cardinality)
- `none` (empty set)
- `univ` (universal set).

```
#{ f: FSObject | f in File + Dir} >= #Dir
#( File + Dir ) >= #Dir
```

1.2.2 On relations

- `->` (cross product)
- `.` (relational join)
- `\~` (transposition)
- `^` (transitive, reflexive closure)
- `<:` (domain restriction, remove all tuples with key in left set)
- `>:` (range restriction, remove all tuples with value in right set)
- `++` (override)

- `iden` (identity relation)
- `[]` (box join: `a[ b ] = b.a`)

```
FSObject in Root.*contents

// r: Root, d1: Dir, d2: Dir, f: File
// contents = {(r, d1), (d1, d2), (d2, f)}
*contents = {(r,d1), (d1,d2), (d2,f), (d1,f), (r,d2), (r,f), (r,r), (d1,d1), (d2,d2),
Root.*contents = {(d1), (d2), (f), (r)} // Take elements on the right which have 'Root'
```

1.2.3 Constraints

- `!` / `not` (negation)
- `&&` / `and` (conjunction)
- `||` / `or` (disjunction)
- `=>` / `implies` (implication)
- `else` (alternative)
- `<=>` / `iff` (equivalence)

```
F => G else H
F implies G else H
(F && G) || ((!F) && H)
(F and G) or ((not F) and H)
```

- `some e` (e has at least one tuple)
- `no e` (e has no tuples)
- `lone e` (e has at most one tuple)
- `one e` (e has exactly one tuple)

```
no Root.parent
```

1.2.4 Quantification

- `all x: e | F` (F holds for every x in e)
- `some x: e | F` (F holds for at least one x in e)
- `no x: e | F` (F holds for no x in e)
- `lone x: e | F` (F holds for at most one x in e)
- `one x: e | F` (F holds for exactly one x in e)

```
all x: e1, y: e2 | F
all disj x, y: e | F

no d: Dir | d in d.^contents // Contents relation is acyclic
```

1.2.5 Multiplicities

- `lone` (empty set or sigleton)
- `one` (sigleton set, default for fields)
- `set` (any set, default for signatures)
- `some` (non-empty seju

1.3 Predicates & Functions

```
// "returns" a boolean
pred isLeave[ f: FSObject ] {
  f in File || no f.contents
}

// "returns" anything else
fun leaves[ f: FSObject ]: set FSObject {
  { x: f.*contents | isLeave[ x ] }
}
```

1.3.1 Analyzer

You can tell the analyzer to search for instances of satisfying functions/predicates with the `run` command:

```
run isLeave
run isLeave for 5
run isLeave for 5 Dir, 2 File
run isLeave for exactly 5 Dir
run isLeave for 5 but 3 Dir
run isLeave for 5 but exactly 3 Dir
```

1.3.2 Facts

Facts are constraints that always hold.

```
fact { all d: Dir, o: d.contents | o.parent = d }
fact { no d: Dir | d in d.^contents }

// Can be after signature
sig Array {
  length: Int
  data: { i: Int | 0 <= i && i < length } -> lone E
} {
  0 <= length
}
```

1.3.3 Checking the model

Assertions aren't enforced, but rather tested by the analyzer. Prefer them over facts.

```
assert nonEmptyRoot { !isLeave[ Root ] }
assert acyclic { no d: Dir | d in d.^contents }

check nonEmptyRoot for 3
check acyclic for 5
```

1.4 Libraries

```
open util/boolean
// Enables the Bool signature, check with isTrue/isFalse

open util/ordering
```