Software Engineering Mathematics and Specification

Basic Set Theory in Alloy

Sets

A *set* represents a collection of items.

Alloy provides a few built-in sets:

- **univ** is the universal set: it contains all items present in the current model
- **none** is the empty set: it contains nothing
- **Int** is the set of integers

[Technical aside: integers in Alloy are represented in **2's complement** notation, with a default *bit width* of 4. This means there are 16 possible values, from -8 to 7.]

Adding our own sets

- A *signature* introduces a new set, e.g.

  sig Fruit { }

- An *enumeration* introduces a new set containing specific new items, e.g.

  enum Vegetable {tomato, lettuce, celery, pea, carrot}

Multiplicities

- When we add a set, we can specify how many members it contains by adding a *multiplicity* constraint:

```
one sig Fruit { }      // exactly one member

lone sig Fruit { }      // "less or one" member, i.e., 0 or 1

some sig Fruit { }    //  at least one member, i.e., 1 or more
```

- If we want to specify a different size, we can use a *fact*:

```
sig Fruit { }

fact FiveFruit { # Fruit = 5 }      // # means "the number of"
```

Creating subsets of a set

- A set can be introduced as a subset of another set. There are two ways of doing this:

  sig Apple, Banana, Pear **extends** Fruit { }

  sig Fresh, Expensive **in** Fruit { }

- **extends** introduces subsets which are mutually disjoint (do not have members in common).

- **in** introduces subsets which may overlap.

Abstract signatures

- An *abstract* signature introduces a set that contains nothing apart from the members of sets that extend that signature:

```
abstract sig Fruit { }

sig Apple, Banana, Pear extends Fruit { }
```

Operations on sets

+ union

  Something is in s + t when it is in s **or** in t (or both)

& intersection

  Something is in s & t when it is in **both** s **and** t

- difference

  Something is in s – t when it is in s **but not** in t

number or cardinality

  # s is the number of members in s

comprehension

create a set from a logical property defining its members, e.g:

{ i:**Int** | i > 5 }   // the set of integers greater than 5

Logical expressions using sets

in subset

s **in** t is true if every member of s is also in t

in membership

a **in** s is true if a is a member of s

= equality

s = t is true if s and t have exactly the same members

some non-emptiness

some s is true if s has at least one member

no emptiness

no p is true if p has no members

Properties of set operations

The set operations have many basic mathematical properties, such as those shown. You can use Alloy to check for yourself that these properties are true in specific examples. (Alloy cannot be used to prove that they hold in all cases – that requires a mathematical proof.)

- General laws

  A + A = A

  A & A = A

  A - A = none

- Commutative laws:

  A + B = B + A

  A & B = B & A

- Associative laws:

  A + (B + C) = (A + B) + C

  A & (B & C) = (A & B) & C

- Distributive laws:

  A + (B & C) = (A + B) & (A + C)

  A & (B + C) = (A & B) + (A & C)