22c:181 / 55:181 Formal Methods in Software Engineering

Sets and Relations

These Notes

- review the concepts of sets and relations required for working with Alloy

- focus on the kind of set operation and definitions used in specifications

- give some small examples of how we will use sets in specifications

Set

- Collection of distinct objects
- Each set's objects are drawn from a larger *domain* of objects all of which have the same type --- sets are homogeneous
- Examples:

domain

{2,4,5,6,...}	set of integers
{red, yellow, blue}	set of colors
{true, false}	set of boolean values
{red, true, 2}	for us, not a set!

Value of a Set

- Is the collection of its members

- Two sets A and B are equal if
  - every member of A is a member of B
  - every member of B is a member of A

- $x \in S$ denotes "x is a member of S"

Defining Sets

- We can define a set by *enumeration*
  - PrimaryColors == {red, yellow, blue}
  - Boolean == {true,false}
  - Evens == {…,-4,-2,0,2,4,…}

- This works fine for finite sets, but
  - what do we mean by "…" ?
  - remember we want to be precise

Defining Sets

- We can define a set by *comprehension*, that is, by describing a property that its elements must share
- Notation:
  - { x : S | P(x) }
  - Form a new set of elements drawn from set/domain S including exactly the elements that satisfy predicate (i.e., Boolean function) P
- Examples:

{ x : N | x < 10} *Naturals less than 10*

{ x : Z | (∃ y : Z | x = 2y) } *Even integers*

{ x : N | false } *Empty set of natural numbers*

Cardinality

- The *Cardinality* (#) of a finite set is the number of its elements
- Examples:
  - # {red, yellow, blue} = 3
  - # {1, 23} = 2
  - # Z = ?
- Cardinalities are defined for infinite sets too, but we'll be most concerned with the cardinality of finite sets.

Set Operations

- Union:
  - $X \cup Y \equiv \{e \mid e \in X \text{ or } e \in Y\}$
  - {red} ∪ {blue} = {red, blue}
- Intersection
  - $X \cap Y \equiv \{e \mid e \in X \text{ and } e \in Y\}$
  - {red, blue} ∩ {blue, yellow} = {blue}
- Difference
  - $X \setminus Y \equiv \{e \mid e \in X \text{ and } e \notin Y\}$
  - {red, yellow, blue} \ {blue, yellow} = {red}

Subsets

- A *subset* holds elements drawn from another set
  - $X \subseteq Y$ iff $(\forall\ e \mid e \in X \Rightarrow e \in Y)$
  - $\{1, 7, 17, 24\} \subseteq Z$
- A *proper subset* is a non-equal subset
- Another view of set equality
  - $A = B$ iff $(A \subseteq B \wedge B \subseteq A)$

Power Sets

- The power set of set S (denoted *Pow*(S)) is the set of all subsets of S, i.e.,

$$Pow(S) \equiv \{e \mid e \subseteq S\}$$

- Example:
  - *Pow*({a,b,c}) = {Ø, {a}, {b}, {c},
    {a,b}, {a,c}, {b,c},
    {a,b,c}}

Note: for any S, $\emptyset \subseteq S$ and thus $\emptyset \in Pow(S)$

Exercises

- These slides include questions that you should be able to solve at this point

- They may require you to think some

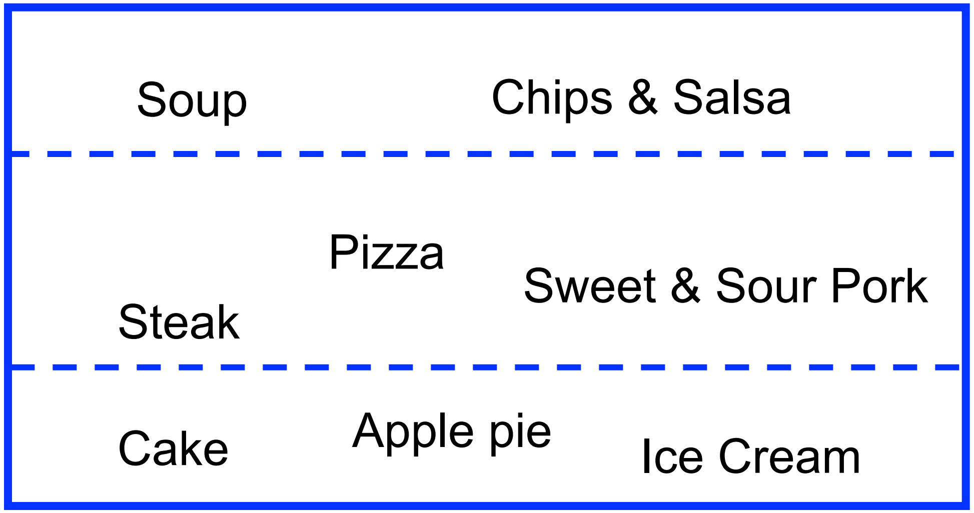
- You should spend some effort in solving them
  - … and may in fact appear on exams

Exercises

- Specifying using comprehension notation
  - Odd positive integers
  - The squares of integers, i.e. {1,4,9,16,…}
- Express the following logic properties on sets without using the # operator
  - Set has at least one element
  - Set has no elements
  - Set has exactly one element
  - Set has at least two elements
  - Set has exactly two elements

Set Partitioning

- Sets are *disjoint* if they share no elements
- Often when modeling, we will take some set S and divide its members into disjoint subsets called *partitions*.
- Each member of S belongs to exactly one partition.



Example

Model residential scenarios

- Basic domains: *Person*, *Residence*

- Partitions:
  - Partition *Person* into *Child*, *Student*, *Adult*
  - Partition *Residence* into *Home*, *DormRoom*, *Apartment*

Exercises

- Express the following properties of pairs of sets
  - Two sets are disjoint
  - Two sets form a partitioning of a third set

Expressing Relationships

- It's useful to be able to refer to structured values
  - a group of values that are bound together
  - e.g., struct, record, object fields
- Alloy is a calculus of *relations*
- All of our Alloy models will be built using relations (sets of tuples).
- … but first some basic definitions

Product

- Given two sets A and B, the product of A and B, usually denoted A x B, is the set of all possible pairs (a, b) where $a \in A$ and $b \in B$.

$$A \times B \equiv \{(a, b) \mid a \in A \text{ and } b \in B\}$$

- Example: PrimaryColor x Boolean:

{ (red,true), (red, false),
(blue,true), (blue, false),
(yellow, true), (yellow, false) }

Relation

- A binary relation R between A and B is an element of *Pow* (A x B), i.e., R ⊆ A x B

- Examples:
  - Parent : Person x Person
    - Parent == {(John, Autumn), (John, Sam)}
  - Square : Z x N
    - Square == {(1,1), (-1,1), (-2,4)}
  - ClassGrades : Person x {A, B, C, D, F}
    - ClassGrades == {(Todd,A), (Jane,B)}

Relation

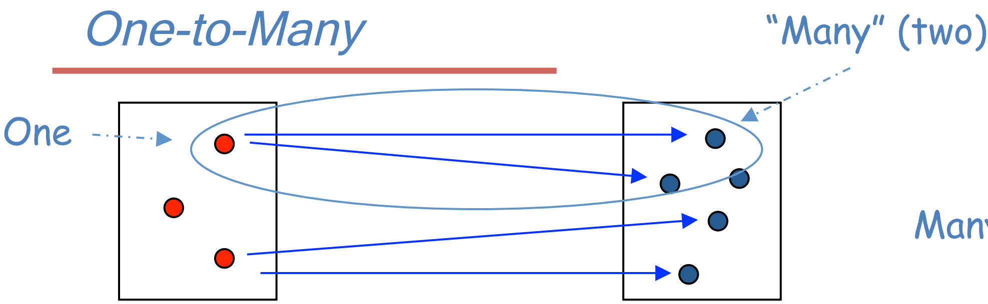
- A ternary relation R between A, B and C is an element of *Pow* (A x B x C)
- Example:
  - FavoriteBeer : Person x Beer x Price
    - FavoriteBeer == {(John, Miller, $2), (Ted, Heineken, $4), (Steve, Miller, $2)}
- N-ary relations with n>3 are defined analogously (n is the arity of the relation)

Binary Relations

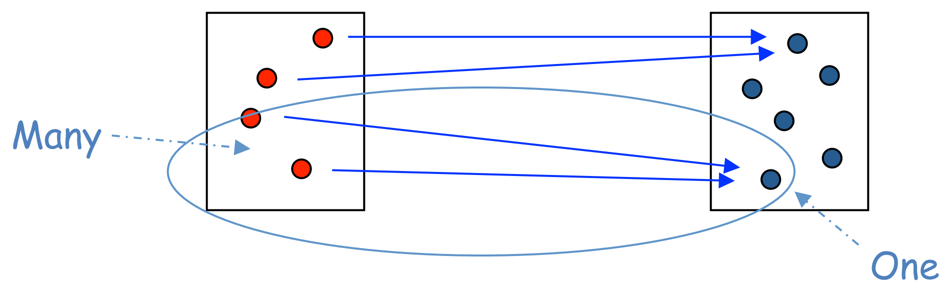
- The set of first elements
  - is the *definition domain* of the relation
  - *domain* (Parent) = {John}   NOT Person!
- The set of last elements
  - is the *image* of the relation
  - *image* (Square) = {1,4}   NOT **N**!
- How about {(1,blue), (2,blue), (1,red)}
  - domain?   image?

Common Relation Structures

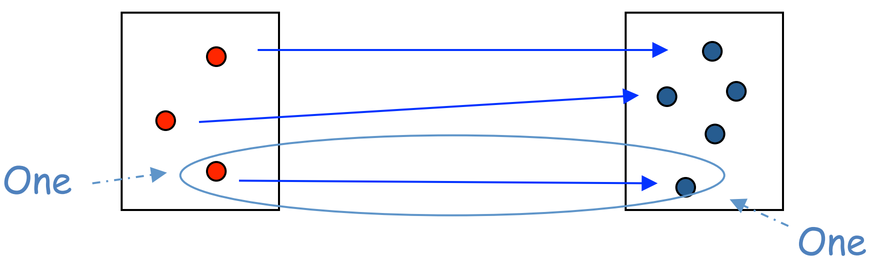
One-to-Many



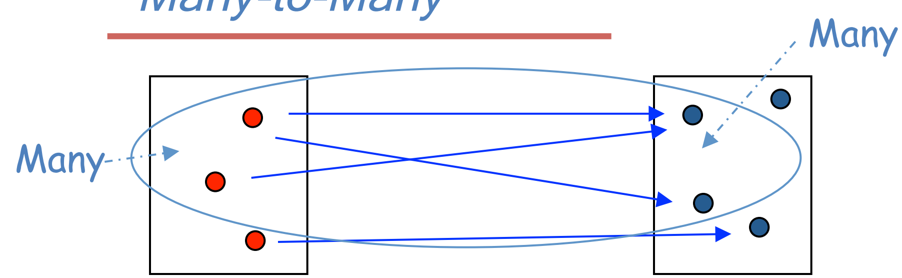
Many-to-One



One-to-One



Many-to-Many



Functions

- A *function* is a relation F of arity n+1 containing no two distinct tuples with the same first n elements, i.e., for n = 1,

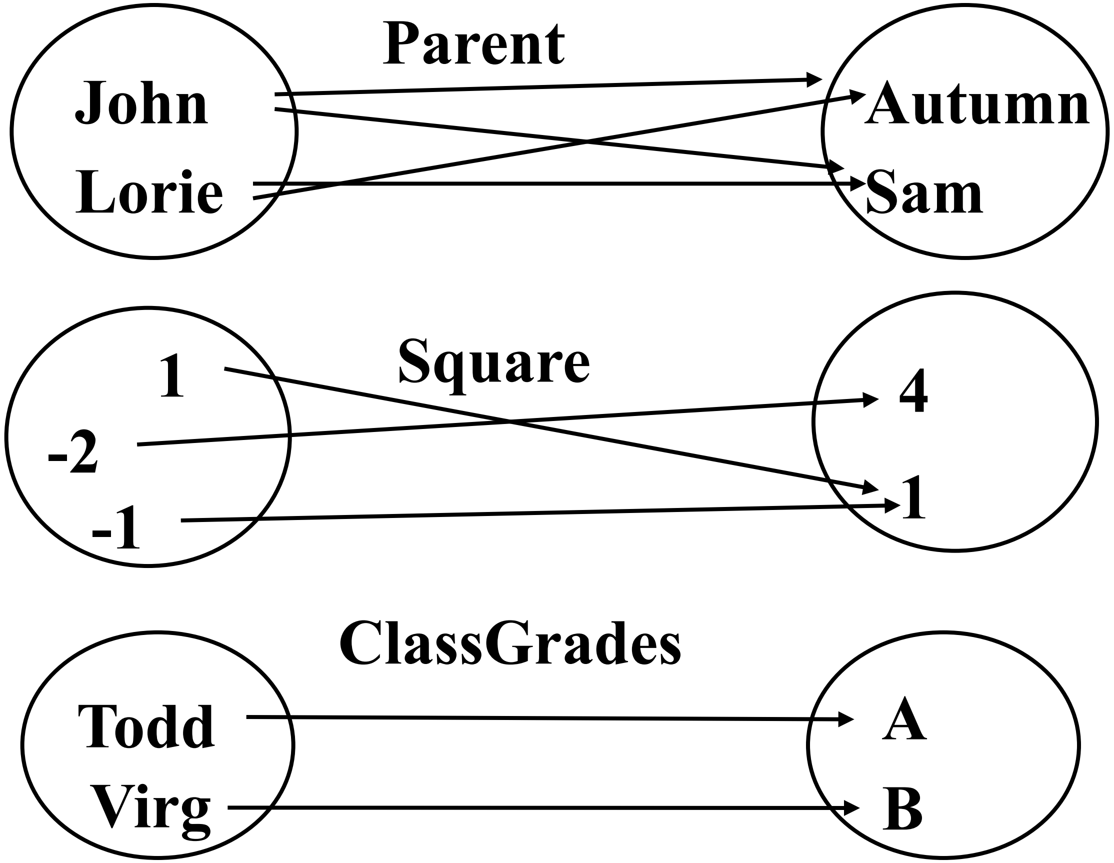
$$\forall (a_1, b_1) \in F, \forall (a_2, b_2) \in F, (a_1 = a_2 \Rightarrow b_1 = b_2)$$

- Examples:
  - {(2, red), (3, blue), (5, red)}
  - {(4, 2), (6,3), (8, 4)}
- Instead of F: A1 x A2 x … x An x B, we write F: A1 x A2 … x An -> B

Exercises

- Which of the following are functions?
  - Parent == {(John,Autumn), (John,Sam)}
  - Square == {(1,1), (-1,1), (-2,4)}
  - ClassGrades == {(Todd,A), (Virg,B)}

Relations vs. Functions



Many-to-many

Many-to-one

One-to-one

In other words, a function is a relation that is X-to-one.

Special Kinds of Functions

- Consider a function f from S to T

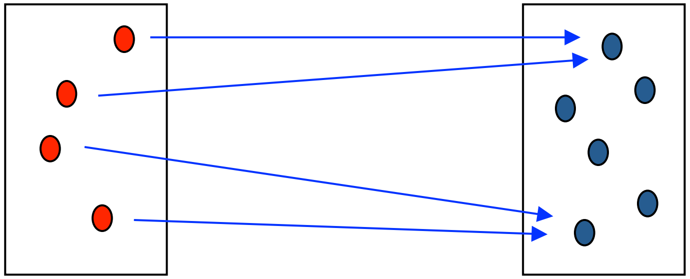
- f is *total* if defined for all values of S
- f is *partial* if defined for some values of S

- Examples
  - Squares : Z -> N, Squares = {(-1,1), (2,4)}
  - Abs = {(x,y) : Z x N |

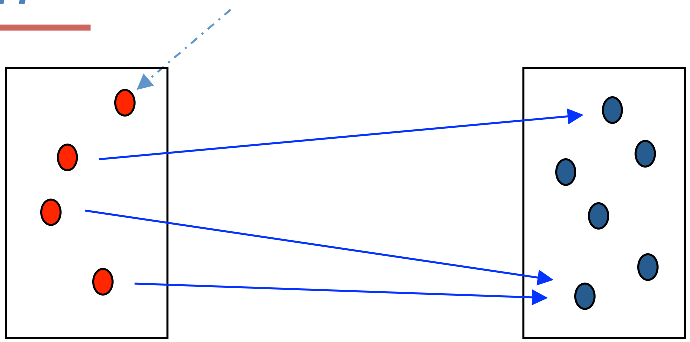
    (x < 0 and y = -x) or (x ≥ 0 and y = x)}

Function Structures

Total Function



Partial Function



<u>Note</u>: the empty relation is a partial function

Special Kinds of Functions

A function f: S -> T is

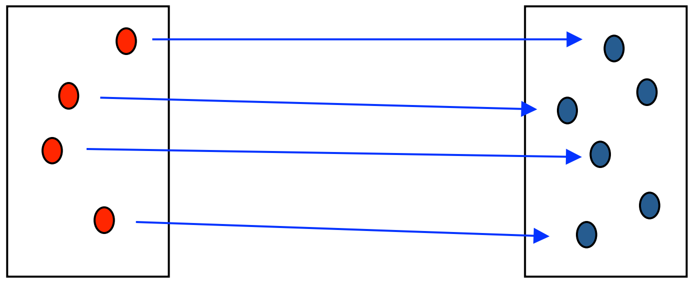
- *injective* (*one-to-one*) if no image element is associated with multiple domain elements
- *surjective* (*onto*) if its image is T
- *Bijective* if it is both injective and surjective

We'll see that these come up frequently

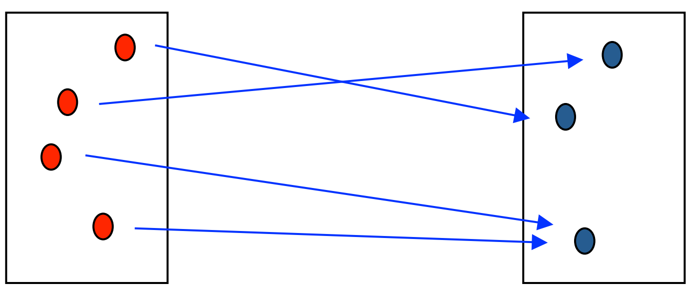
– can be used to define properties concisely

Function Structures

Injective Function



Surjective Function

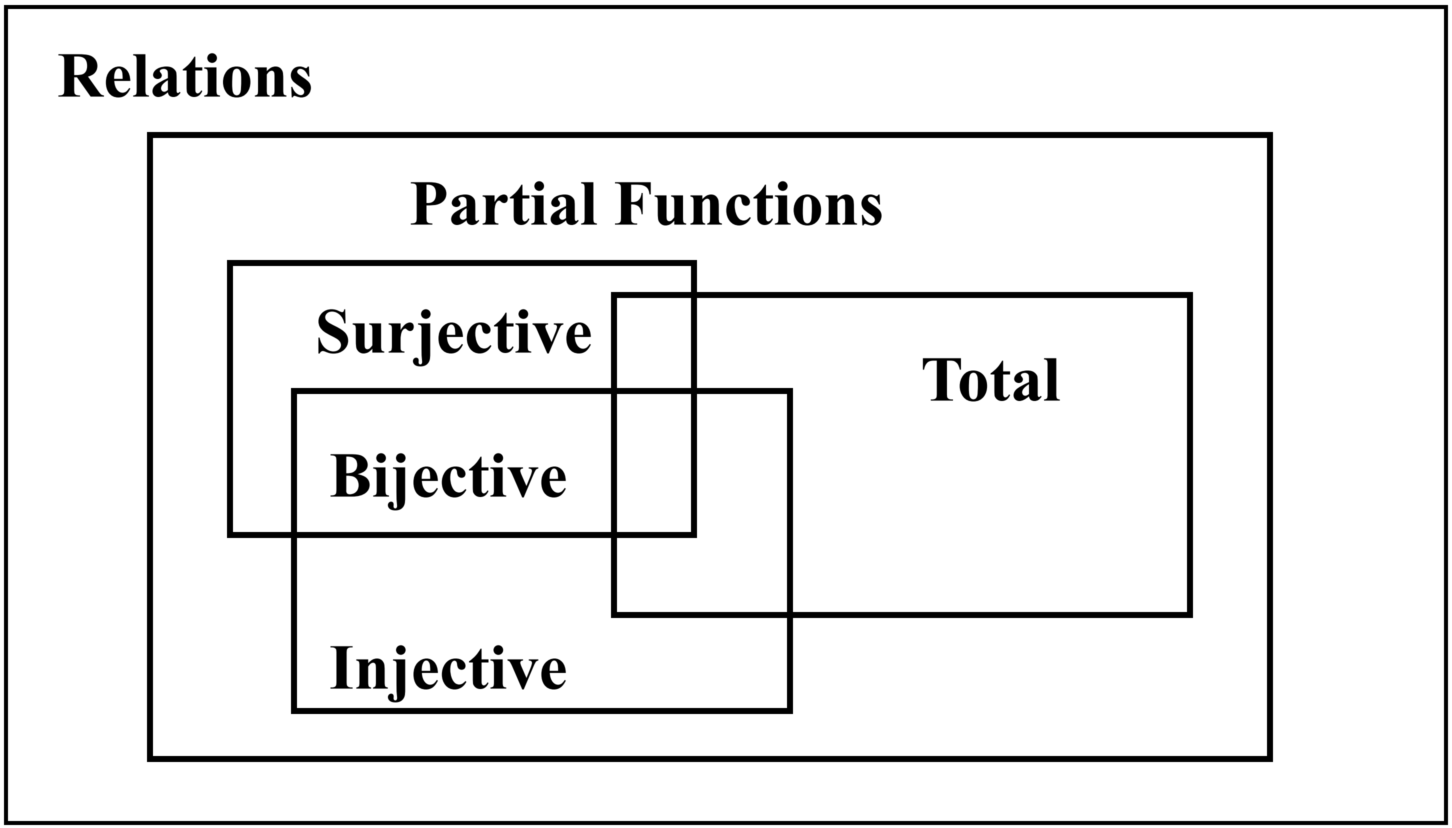


Exercises

- What kind of function/relation is Abs?
  - Abs = {(x,y) : Z x N | (x < 0 and y = -x) or (x ≥ 0 and y = x)}

- How about Squares?
  - Squares : Z x N, Squares = {(-1,1),(2,4)}

Special Cases



Functions as Sets

- Functions are relations and hence sets
- We can apply all of the usual operators
  - ClassGrades == {(Todd,A), (Jane,B)}
  - #(ClassGrades U {(Matt,C)}) = 3

Exercises

- In the following if an operator fails to preserve a property give an example
- What operators preserve function-ness?
  - $\cap$?
  - $\cup$?
  - $\setminus$?
- What operators preserve surjectivity?
- What operators preserve injectivity?

Relation Composition

- Use two relations to produce a new one
  - map domain of first to image of second
  - Given s: A x B and r: B x C then s;r : A x C

$$s;r \equiv \{(a,c) \mid (a,b) \in s \text{ and } (b,c) \in r\}$$

- For example
  - s == {(red,1), (blue,2)}
  - r == {(1,2), (2,4), (3,6)}
  - s;r = {(red,2), (blue,4)}

Relation Closure

- Intuitively, the closure of a relation $r: S \times S$ (written r^+) is what you get when you keep navigating through r until you can't go any farther.

$$r^+ \equiv r \cup (r;r) \cup (r;r;r) \cup \ldots$$

- For example
  - GrandParent == Parent;Parent
  - Ancestor == Parent^+

Relation Transpose

- Intuitively, the transpose of a relation $r: S \times T$ (written $\sim r$) is what you get when you reverse all the pairs in r.

$$\sim r \equiv \{(b,a) \mid (a,b) \in r\}$$

- For example
  - ChildOf == ~Parent
  - DescendantOf == $(\sim\text{Parent})^+$

Exercises

- In the following if an operator fails to preserve a property give an example

- What properties, i.e., function-ness, onto-ness, 1-1-ness, by the relation operators?
  - composition (;)
  - closure ($^+$)
  - transpose (~)

Acknowledgements

- Some of these slides are adapted from
  - David Garlan’s slides from Lecture 3 of his course of Software Models entitled “Sets, Relations, and Functions” (http://www.cs.cmu.edu/afs/cs/academic/class/15671-f97/www/)