CS:5810 Formal Methods in Software Engineering

Sets and Relations

These Notes

- review the concepts of sets and relations required to work with the Alloy language
- focus on the kind of set operation and definitions used in specifications
- give some small examples of how we will use sets in specifications

Sets

A set is a *collection* of distinct objects

A 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!

Set Values

The value of a set is the collection of its members

Two sets A and B are equal iff

– every member of A is a member of B

– every member of B is a member of A

Notation:

- $x \in S$ denotes "x is a member of S"
- $\emptyset$ denotes the empty set

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: $\{ x : D \mid P(x) \}$

Form a new set of elements drawn from domain D by including exactly the elements that satisfy predicate (i.e., Boolean function) P

Examples:

$\{ x : \mathrm{N} \mid x < 10\}$ *Naturals less than 10*

$\{ x : \mathrm{Z} \mid (\exists\, y : \mathrm{Z} \mid x = 2y) \}$ *Even integers*

$\{ x : \mathrm{N} \mid x > x \}$ *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
- # $\mathbb{Z}$ = ?

Cardinalities are defined for infinite sets too but we'll be mostly concerned with the cardinality of finite sets

Set Operations

Union (X, Y sets over domain D): $X \cup Y \equiv \{ e: D \mid e \in X \text{ or } e \in Y \}$

– {red} ∪ {blue} = {red, blue}

Intersection: $X \cap Y \equiv \{ e: D \mid e \in X \text{ and } e \in Y \}$

– {red, blue} ∩ {blue, yellow} = {blue}

Difference $X \setminus Y \equiv \{ e: D \mid e \in X \text{ and } e \notin Y \}$

– {red, yellow, blue} \ {blue, yellow} = {red}

Subsets

A *subset* holds elements drawn from another set

$X \subseteq Y$ iff every element of X is in Y

Example: $\{ 1, 7, 24 \} \subseteq \{ 1, 7, 17, 24 \} \subseteq \mathbb{Z}$

A *proper subset* is a non-equal subset

Another view of set equality: $A = B$ iff ($A \subseteq B$ and $B \subseteq A$)

Power Sets

The power set of set S, denoted *Pow* (S), is the set of all subsets of S:

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

Example:

$Pow(\{a,b,c\}) = \{ \emptyset, \{a\}, \{b\}, \{c\}, \{a,b\}, \{a,c\}, \{b,c\}, \{a,b,c\} \}$

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

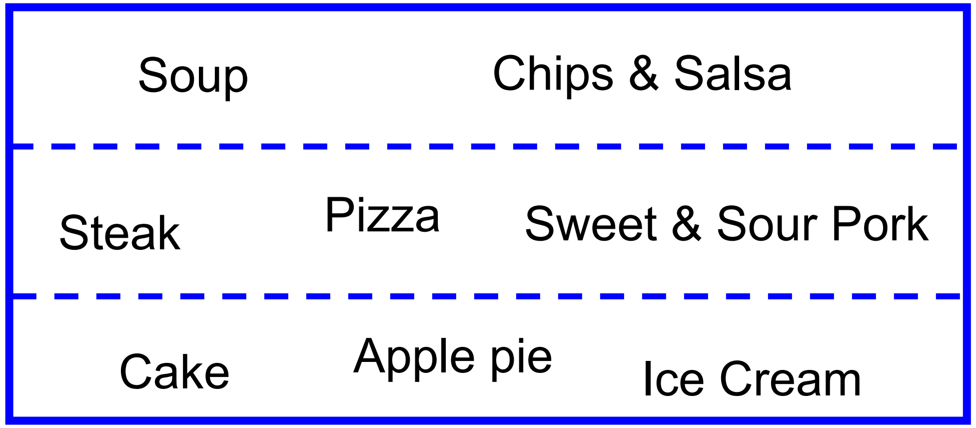
You should spend some effort in solving them
… and may in fact appear on exams

Exercises

1. Specifying using comprehension notation
   a) Odd positive integers
   b) The squares of integers, i.e. { 0, 1, 4, 9, 16,… }
2. Express the following logic properties on sets without using the # operator
   a) Set has at least one element
   b) Set has no elements
   c) Set has exactly one element
   d) Set has at least two elements
   e) Set has exactly two elements

Set Partitioning

- Sets are *disjoint* if they share no elements
- We will often take some set S and divide its members into disjoint subsets called *blocks* or *parts*
- We call this division a *partition*
- Each member of S belongs to exactly one block of the partition



Partition Example

Model residential scenarios

Basic domains: *Person*, *Residence*

Partitions:

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

Exercises

1. Express the following properties of pairs of sets
   a) Two sets are disjoint
   b) 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* (sets of tuples)

All of our Alloy models will be built using relations

… but first some basic definitions

Products

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

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

Example:

$$\text{PrimaryColor} \times \text{Boolean} = \left\{ \begin{array}{ll} (\text{red}, \text{true}), & (\text{red}, \text{false}), \\ (\text{blue}, \text{true}), & (\text{blue}, \text{false}), \\ (\text{yellow}, \text{true}), & (\text{yellow}, \text{false}) \end{array} \right\}$$

Binary Relations

A binary relation R between A and B is an element of *Pow* (A x B), i.e., $R \subseteq A \times B$

A x B

domain *co-domain*

Examples:

Parent : Person x Person = { (John, June), (John, Sam) }

Square : Z x N = { (1, 1), (-1, 1), (-2, 4) }

ClassGrades : Person x { A, B, C, D, F } = { (Kim, A), (Alex, B) }

Binary Relations

The set of first elements is the *definition domain* of the relation

- Parent = { (John, Autumn), (John, Sam) }
- *defdomain* (Parent) = { John }   not Person!

The set of second elements is the *image* of the relation

- *image* (Square) = { 1, 4 }   not $\mathbb{N}$!

What about { (1,blue), (2,blue), (1,red) }

- definition domain?   image?

N-ary Relations

A ternary relation R between A, B and C is an element of *Pow* (A x B x C)

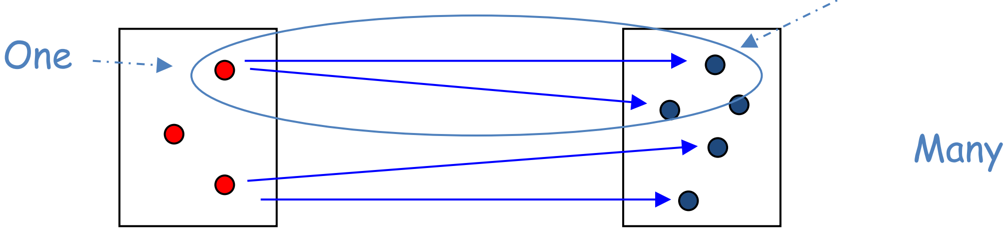
Example:

FavoriteBeer : Person x Beer x Price
= { (John, Miller, \$2), (Ted, Heineken, \$4), (Steve, Miller, \$2) }

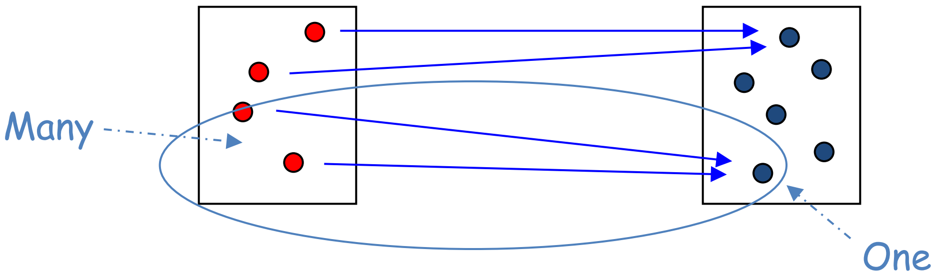
N-ary relations with n>3 are defined analogously
(n is the arity of the relation)

Common Relation Structures

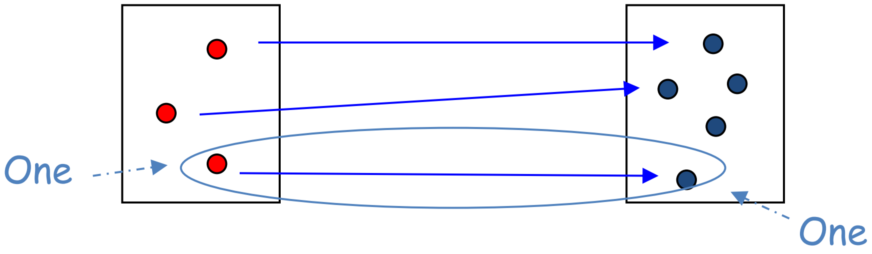
One-to-Many



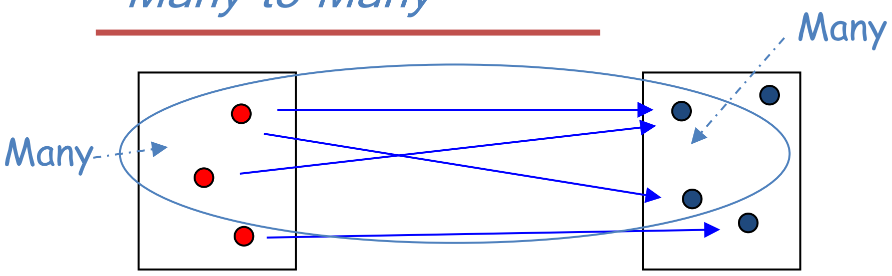
Many-to-One



One-to-One



Many-to-Many



Functional Relations

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, $\nexists\ (a, b_1), (a, b_2) \in F$ s.t. $b_1 \neq b_2$

Examples:

- { (2, red), (3, blue), (5, red) } ✓
- { (4, 2), (6, 3), (8, 4) } ✓
- { (2, red), (3, blue), (2, blue) } ✗

Instead of $F: A_1 \times A_2 \times \ldots \times A_n \times B$ we write $F: A_1 \times A_2 \times \ldots \times A_n \text{ -> } B$

Exercises

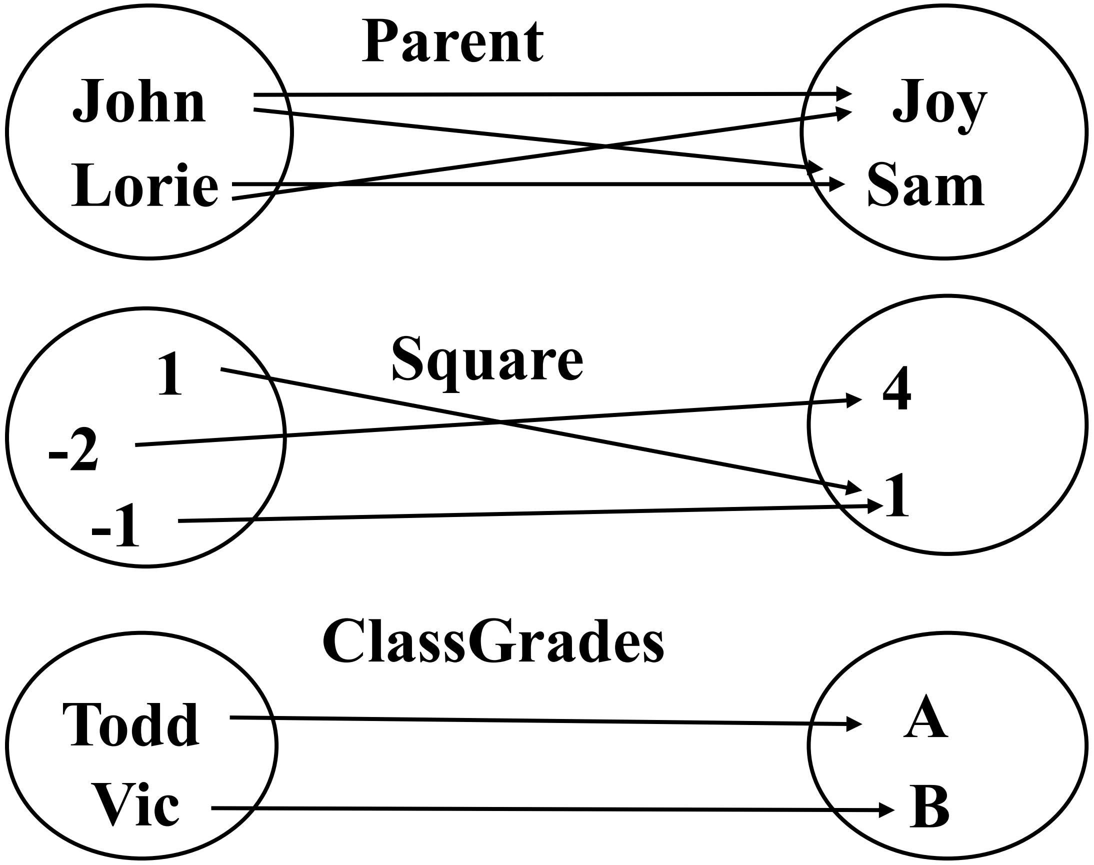
Which of the following are functions?

1. Parent = { (John, Ann), (John, Sam), (Sam, Joy) }

2. Square = { (1, 1), (-1, 1), (-2, 4) }

3. ClassGrades = { (Todd, A), (Vic, B) }

Relations vs. Functions



Many-to-many

Many-to-one

One-to-one

A function is an X-to-one relation

Special Kinds of Functions

Consider a function f from S to T

f is *total* if defined for all values of S

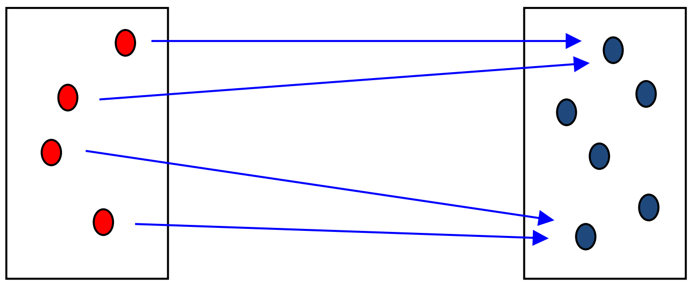
f is *partial* if undefined for some values of S

Examples:

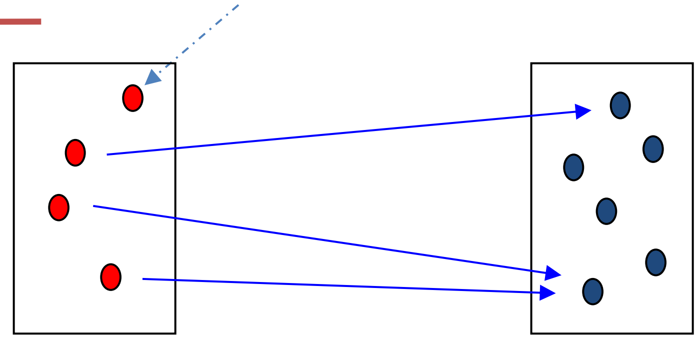
- Squares : Z -> N = {..., (-1,1), (0,0), (1, 1), (2,4), ...} total
- SquareRoot : N -> N = { (x, y) : N x N | y^2 = x) } partial

Function Structures

Total Function



Partial Function



Note: the empty relation over a non-empty domain is a partial function

Special Kinds of Functions

A function f: S -> T is

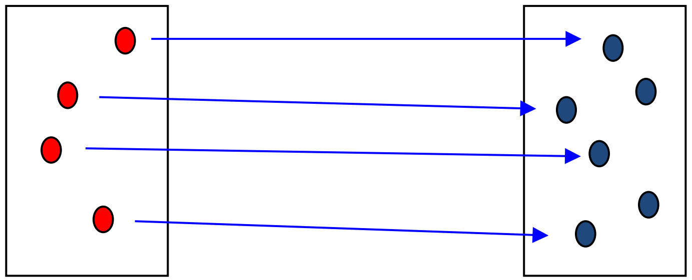
- *injective* (*one-to-one*) if no image element is associated with multiple domain elements
- *surjective* (*onto*) if its image is T
- *bijective* if it is both injective and surjective

We'll see that these come up frequently

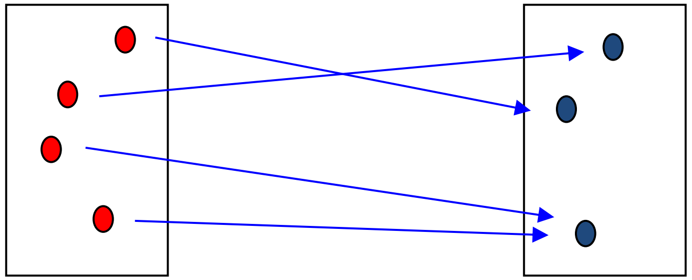
– can be used to define properties concisely

Function Structures

Injective Function



Surjective Function



Exercises

1. What kind of function/relation is Abs?

   Abs : $\mathbb{Z} \times \mathbb{N}$ = { (x, y) : $\mathbb{Z} \times \mathbb{N}$ | (x < 0 and y = -x) or (x ≥ 0 and y = x) }

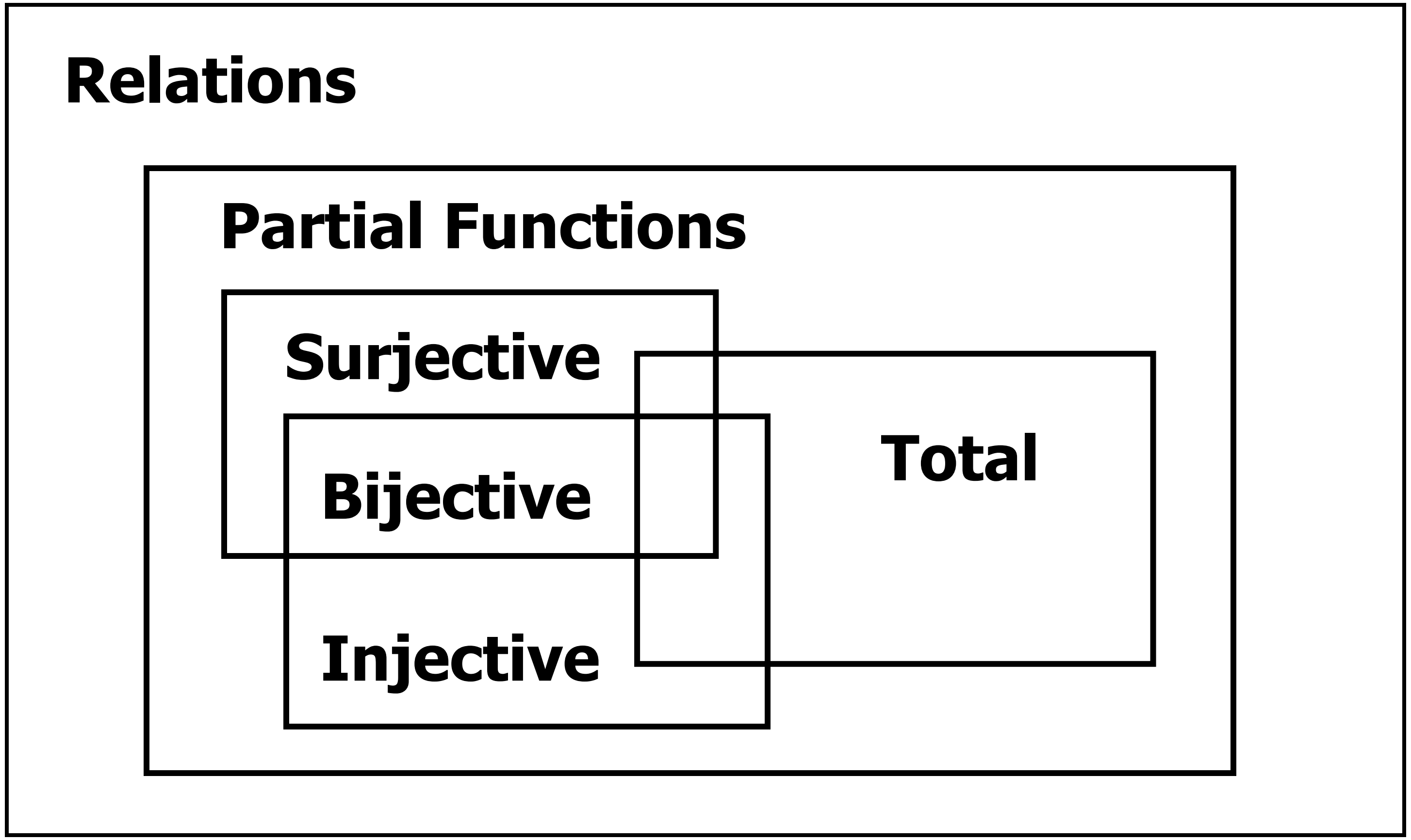
2. How about Squares?

   Squares : $\mathbb{Z} \times \mathbb{N}$ = { (x, y) : $\mathbb{Z} \times \mathbb{N}$ | y = x·x }

3. How about Rel?

   Rel : $\mathbb{Z} \times \mathbb{N}$ = { (x, y) : $\mathbb{Z} \times \mathbb{N}$ | y = 2·x if x >= 0,
   y = 2·(-x) - 1 if x < 0 }

Special Cases



Functions as Sets

Functions are relations and hence sets

We can apply to them all the usual operators

- ClassGrades = { (Todd, A), (Jane, B) }
- #(ClassGrades ∪ { (Matt, C) }) = 3

Exercises

1. In the following if an operator fails to preserve a property give an example
2. What operators preserve function-ness?
   a) $\cap$?
   b) $\cup$?
   c) $\setminus$?
3. What operators preserve surjectivity?
4. 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

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

Example:

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

Not limited to binary relations

Relation Transitive Closure

Intuitively, the *transitive closure* r^+ of a binary relation $r: S \times S$ is the result of adding a direct link (a,b) to r for every a and b where b is reachable from a along r :

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

Formally, $r^+ \equiv$ smallest transitive relation containing r

Example:

- GrandParent = Parent ; Parent
- GrandGrandParent = Parent ; GrandParent
- Ancestor = Parent ∪ GrandParent ∪ GrandGrandParent ∪ … = Parent^+

Relation Transpose

Intuitively, the *transpose* ~r of a relation r: S x T is the relation obtained by reversing all the pairs in r

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

Example:

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

Exercises

1. What properties, i.e., function-ness, onto-ness, 1-1-ness, are preserved by these relation operators?
   a) composition (;)
   b) closure ($^+$)
   c) transpose (~)

2. If an operator fails to preserve a property give an example

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/)