CIS 771: Software Specifications

Lecture 2: Sets and Relations

This Lecture…

- …reviews the concepts of sets and relations required for working Alloy, UML's OCL.
- …focuses on the forms of set operation and definitions used in specifications.
- …gives some brief 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:

{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!

domain

Value of a Set

- Is its membership
- 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 ∈ 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

- *Cardinality* is the size of a set
- Examples:
  - # {red,yellow,blue} = 3
  - # {1,2,2} = 2
  - # Z
- Don't worry about infinite sets too much. We'll be using them in a very practical way

Set Operations

- Union:
  - $X \cup Y \equiv \{e \mid e \in X \vee e \in Y\}$
  - $\{red\} \cup \{blue\} = \{red,blue\}$
- Intersection
  - $X \cap Y \equiv \{e \mid e \in X \wedge e \in Y\}$
  - $\{red,blue\} \cap \{blue,yellow\} = \{blue\}$
- Difference
  - $X \setminus Y \equiv \{e \mid e \in X \wedge e \notin Y\}$
  - $\{red,yellow,blue\} \setminus \{blue,yellow\} = \{red\}$

Subsets

- A *subset* holds elements drawn from another set
  - $X \subseteq Y \equiv (\forall e \mid e \in X \rightarrow e \in Y)$
  - $\{1,7,17,23\} \subseteq Z$
- A *proper subset* is a non-equal subset
- Another view of equality
  - $A = B \equiv (A \subseteq B \wedge B \subseteq A)$

Power Sets

- The power set of set S (denoted $\mathbf{P}(S)$) is the set of all subsets of S, i.e.,

$$\mathbf{P}(S) \equiv \{e \mid e \subseteq S\}$$

- Example:
  - $\mathbf{P}(\{a,b,c\}) = \{\varnothing, \{a\}, \{b\}, \{c\}, \{a,b\}, \{a,c\}, \{b,c\}, \{a,b,c\}\}$

Note: for any S, $\varnothing \subseteq S$ and thus $\varnothing \in \mathbf{P}(S)$

For You To Do

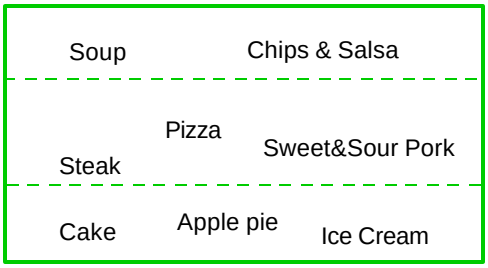
- These slides include questions that you should be able to solve at this point
- They may require you to think some
- You should spend some effort in solving them
- They are representative of quiz problems
  - ... and may in fact appear on weekly quizzes or exams
- Pause the lecture, solve the problems, and restart the lecture

For you to do **(pause here)**

- 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: *Persons*, *Residences*
- Partitions:
  - Partition *Persons* into *Child*, *Student*, *Adult*
  - Partition *Residences* into *Home*, *DormRoom*, *Apartment*, *Shelter*

For you to do (pause here)

- 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 *binary relations*
  - UML's OCL is also a relational formalism
- All of our Alloy models will be built up using binary relations (sets of pairs).
- ... but first some basic definitions

Product

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

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

- Example: PrimaryColor x Boolean:

$$\left\{ \begin{array}{ll} (red,true), & (red, false), \\ (blue,true), & (blue, false), \\ (yellow, true), & (yellow, false) \end{array} \right\}$$

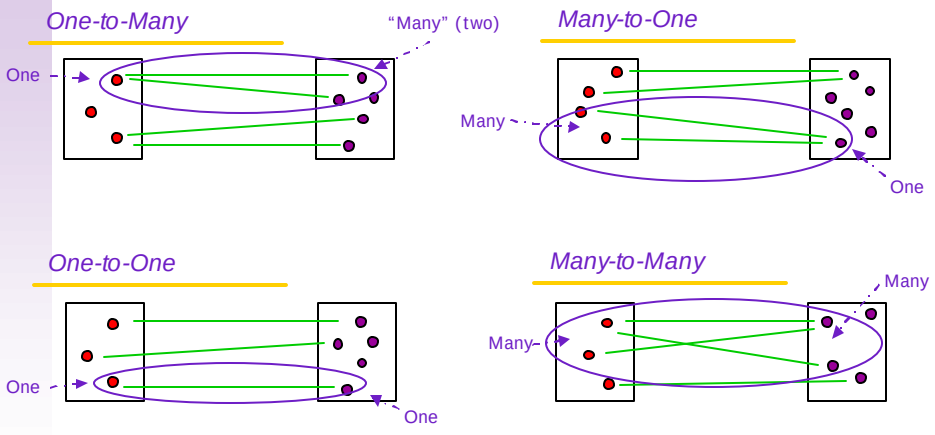
Relation

- A binary relation R between A and B is an element of $\mathbb{P}(A \times B)$, i.e., $R \subseteq A \times B$
- 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),(Virg,B)}

Tuple Components

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

Common Relation Structures



Functions

- A *function* is a relation F such that no two distinct pairs contain the same first element, i.e.,

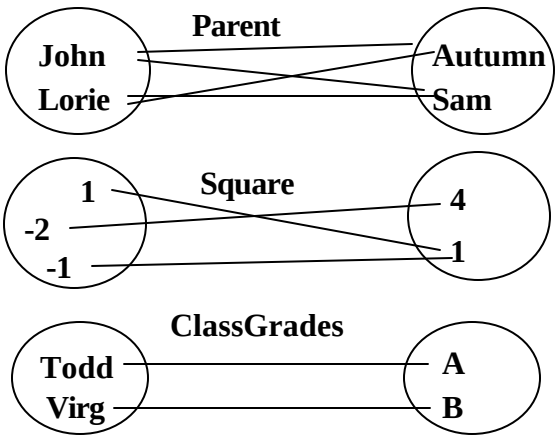
$$\forall(a1,b1)\in F,\forall(a2,b2)\in F,a1=a2\rightarrow b1=b2$$

- Examples:
  - e.g., {(2,red),(3,blue),(5,red)}
  - e.g., {(4,2),(6,3),(8,4)}

For You To Do (pause here)

- 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

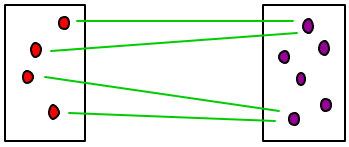
In other words, a function is a relation that is XXXX-to-one.

Special Kinds of Functions

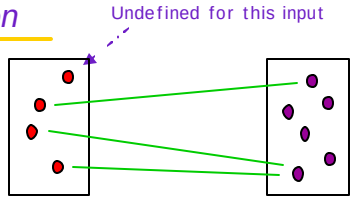
- Consider a function f from S to T
- f is *total* if defined for all values of S
- f is *partial* if defined for some values of S
- Examples
  - Squares : Z => N, Squares = {(-1,1),(2,4)}
  - Abs = {(x,y) : Z x N |
    (x < 0 and y=-x) or (x>=0 and y=x)}

Function Structures

Total Function



Partial Function



Note: the empty relation is a partial function

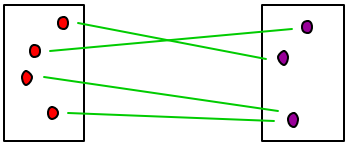
Special Kinds of Functions

- f is *one-to-one* (*injective*) if no image element is associated with multiple ddomain elements
- f is *onto* (*surjective*) if its image is T
- if f is both of these we call it *bijective*
- We'll see that these come up frequently
  - can be used to define properties concisely

Function Structures

Injective Function

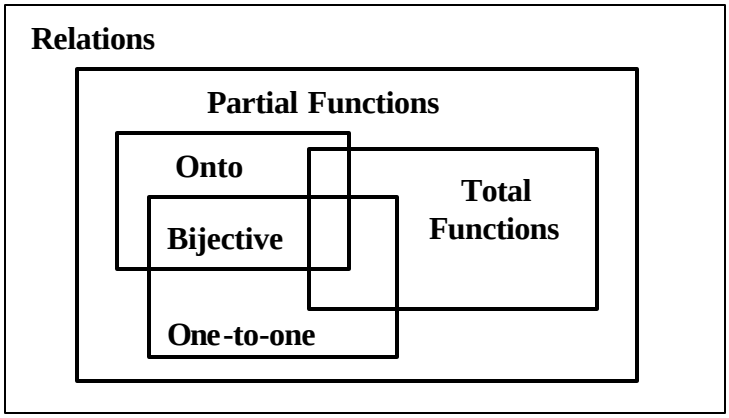
Surjective Function



For You To Do **(pause here)**

- 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),(Virg,B)}
  - #(ClassGrades ∪ {(Matt,C)}) = 3

For You To Do (pause here)

- In the following if an operator fails to preserve a property give an example
- What operators preserve function-ness?
  - $\cap$?
  - $\cup$?
  - \ ?
- What operators preserve onto-ness?
- What operators preserve 1-1-ness?

Relation Composition

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

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

- 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 x 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 x T (written ~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 == +~Parent

For You To Do (pause here)

- 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

- A significant portion 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/)