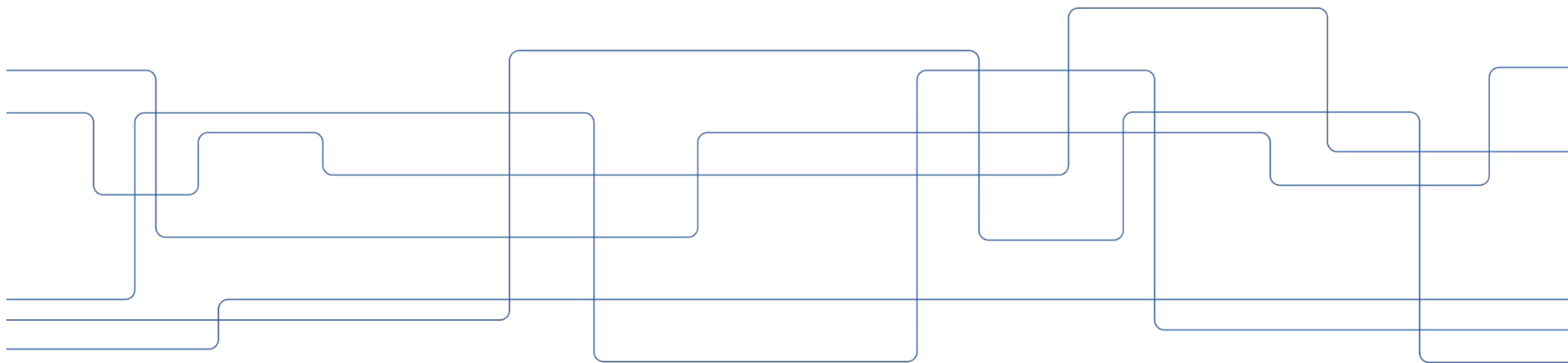




DD2460 Lecture 5.

Using Set Theory, Functions and Relations in Formal Specification

Elena Troubitsyna





Lecture outline

Basic operations of sets

Predicates

Operations on relations and functions

Examples of using relations and functions in specification of access control



Basic set theory

- A **set** is a collection of elements.
- Elements of a set may be numbers, names, identifiers, etc.
 - E.g. the set \mathbb{N} is the collections of all natural numbers.
- **Examples:**
 - $\{3, 5, 7, \dots\}$
 - $\{\text{red}, \text{green}, \text{black}\}$
 - $\{\text{yes}, \text{no}\}$
 - $\{\text{wait}, \text{start}, \text{process}, \text{stop}\}$
 - But **not:** $\{1, 2, \text{green}\}$
- Elements of a set are not ordered.
- Set may be finite or infinite.

Membership

- Relationship between an element and a set: is the element a **member** of the set or not?
- For element x and set S , we express the membership relation as follows

$$x \in S \quad ('x \text{ is a member of } S')$$

where \in is a predicate over sets and elements

- **Set membership** is a boolean property relating an element and a set, i.e., either x is in S or x is not in S .
- This means that there is no concept of an element occurring more than once in a set, e.g.,
 - $\{a, a, b, c\} = \{a, b, c\}$;
 - $\{3, 7\} = \{3, 7, 7\}$
- Conversely, the element is not a member of the set: $x \notin S$

Set definition

- If a set has only finite number of elements, then it can be written explicitly, by listing all of its elements within set brackets '{' and '}':
 - **LectureHall** = {1A, 1B, 1C, 1D}
 - **SEMESTRS** = {spring, fall}
- Some sets have predefined names:
 - \mathbb{N} – *the set of natural numbers* {0, 1, 2, 3, ...}
 - \mathbb{Z} – *the set of integers* {... - 2, -1, 0, 1, 2, ...}
- The empty set \emptyset contains no elements at all. It is the **smallest** possible set.

Set comprehension

- Enumerating all of the elements of a set is not always possible.
- Would like to describe a set by in terms of a distinguishing property of its elements.
- Set can be defined by means of a set comprehension:

$$\{ \boldsymbol{x} \mid \boldsymbol{x} \in \boldsymbol{T} \wedge \boldsymbol{P}(\boldsymbol{x}) \}$$

A variable ranging over ... condition

“Set of all x in T that satisfy $P(x)$ ”

- Each element of a set satisfies some criterion. Criteria are defined by predicates.

Examples on set comprehension

- Examples:

- Natural numbers less than 10: $\{x \mid x \in \mathbb{N} \wedge x < 10\}$
- Even integers: $\{x \mid x \in \mathbb{Z} \wedge (\exists y. y \in \mathbb{Z} \wedge 2y = x)\}$
- Sometimes it is helpful to specify a “*pattern*” for the elements
 - E.g. $\{2x \mid x \in \mathbb{N} \wedge x^2 \geq 3\}$



More examples on set comprehension

- Examples:
 - What is the set defined by the set comprehension:

$$\{z \mid z \in \mathbb{N} \wedge z < 100 \wedge (\exists m. m \in \mathbb{Z} \wedge m^3 = z)\}?$$



More examples on set comprehension

- Examples:
 - What is the set defined by the set comprehension:

$$\{z \mid z \in \mathbb{N} \wedge z < 100 \wedge (\exists m. m \in \mathbb{Z} \wedge m^3 = z)\}?$$

Answer: $\{1, 8, 27, 64\}$

Subset and equality relations for sets

- A set S is said to be *subset* of set T when every element of S is also an element of T . This is written as follows:

$$S \subseteq T$$

- For example:
 - $\{3, 7\} \subseteq \{1, 2, 3, 5, 7, 9\};$
 - $\{apple, pear\} \subseteq \{apple, banana, pear, grape\}$
 - $\{Jones, White\} \subseteq \{White, Smith, Jones, Jakson\}$
- A set S is said to be equal to set T when $S \subseteq T$ and $T \subseteq S$
$$S = T$$

More examples

Set membership says nothing about the relationship between the elements of a set other than that they are members of the same set.

- the order in which we enumerate a set is not significant, e.g.,
 - $\{a, b, c\} = \{b, a, c\}$;
- there is no concept of an element occurring more than once in a set, e.g.,
 - $\{a, a, b, c\} = \{a, b, c\}$;

These two characteristics distinguish sets from data structures such as **lists** or **arrays** where elements appear in order and the same element may occur multiple times.

Operations on sets (set operators)

- **Union** of S and T: set of elements in either S or T:

$$S \cup T$$

- **Intersection** of S and T: set of elements in both S and T:

$$S \cap T$$

- **Difference** of S and T: set of elements in S but not in T:

$$S \setminus T$$

Examples on Set Operators

○ Union

- $\{1,2\} \cup \{2,3,5\} = \{1,2,3,5\}$
- $\emptyset \cup \{\text{red}, \text{pink}\} = \{\text{red}, \text{pink}\}$

○ Intersection

- $\{\text{apple}, \text{pear}, \text{grape}\} \cap \{\text{pear}, \text{banana}\} = \{\text{pear}\}$
- $\{\text{radish}, \text{onion}, \text{celery}\} \cap \{\text{pumpkin}, \text{tomato}, \text{carrot}\} = \emptyset$
- $\{2,3,5\} \cap \emptyset = \emptyset$

○ Difference

- $\{\text{chess}, \text{tennis}, \text{football}\} \setminus \{\text{tennis}, \text{golf}\} = \{\text{chess}, \text{football}\}$
- $\{\text{pot}, \text{bucket}, \text{basket}\} \setminus \{\text{needle}, \text{scissors}\} = \{\text{pot}, \text{bucket}, \text{basket}\}$
- $\{\text{red}, \text{pink}\} \setminus \emptyset = \{\text{red}, \text{pink}\}$

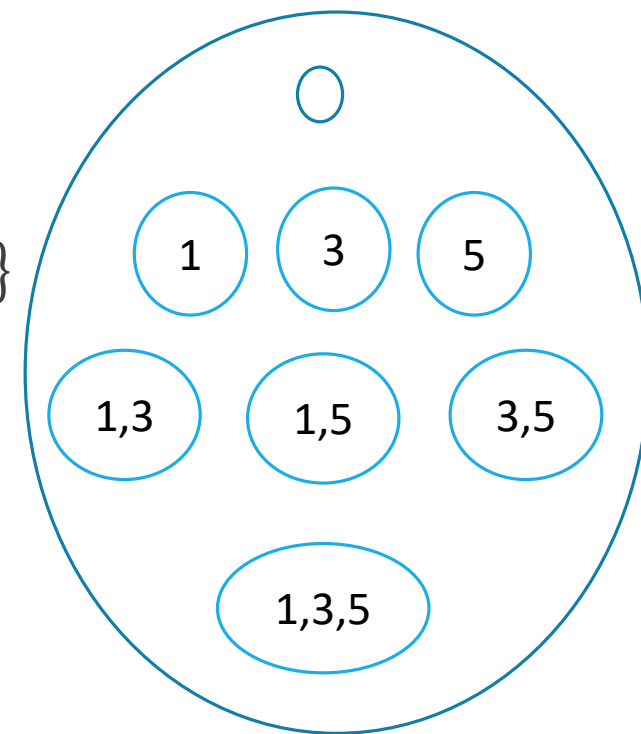
Power sets

- The **power set** of a set S is the set whose elements are all subsets of S ,
written $\mathbb{P}(S)$

- Example,

$$\mathbb{P}(\{1,3,5\}) = \{\emptyset, \{1\}, \{3\}, \{5\}, \{1,3\}, \{1,5\}, \{3,5\}, \{1,3,5\}\}$$

- $S \in \mathbb{P}(T)$ is the same as $S \subseteq T$
- Sets are themselves elements – so we can have **sets of sets**
- Example, $\mathbb{P}(\{1,3,5\})$ is an example of a set of sets





Types of sets

- All the elements of a set must have the same type.
- For example, $\{2, 3, 4\}$ is a set of integers.

$\{2, 3, 4\} \in \mathbb{P}(\mathbb{Z})$.

So the type of $\{2, 3, 4\}$ is $\mathbb{P}(\mathbb{Z})$.

To declare x to be a set of elements of type T we write either

$$x \in \mathbb{P}(T) \quad \text{or} \quad x \subseteq T$$

For example, $\text{math} \subseteq \text{COURCES}$ - so type of math is $\mathbb{P}(\text{COURCES})$



Cardinality

- The number of elements in a set is called its *cardinality*
- In Event-B this is written as **card(S)**
- Examples:
 - $\text{card}(\{1, 2, 3\})=3$
 - $\text{card}(\{a, b, c, d\})=4$
 - $\text{card}(\{\text{Bill}, \text{Anna}, \text{Anna}, \text{Bill}\})=2$
 - $\text{card}(\mathbb{P}(\{1,3,5\}))=8$
- Cardinality is only defined for finite sets.
 - If S is an infinite set, then $\text{card}(S)$ is undefined. Whenever you use the card operator, you must ensure that it is only applied to a finite set.



Expressions

- **Expressions** are syntactic structures for specifying values (elements or sets)
- **Basic** expressions are
 - literals (e.g., 3, \emptyset);
 - variables (e.g., ***x***, ***a***, ***room***, ***registered***);
 - carrier sets (e.g., **S**, **STUDENTS**, **FRUITS**).
- Compound expressions are formed by applying expressions to operators such as

$$x + y \quad \text{and} \quad S \cup T$$

to any level of nesting.



Predicates

- **Predicates** are syntactic structures for specifying logical statements, i.e., statements that are either **TRUE** or **FALSE** (but not both!!!).
- Equality of expressions is an example of a predicate
 - e.g., *registered = registered_spring \cup registered_fall*.
- Set membership, e.g., $5 \in \mathbb{N}$
- Subset relations, e.g., $S \subseteq T$
- For integer elements we can write ordering predicates such as $x < y$.



Predicate logic

- Basic predicates: $x \in S, S \subseteq T, x \leq y$
- Predicate operators:

Name	Predicate	Definitions
Negation	$\neg P$	P does not hold
Conjunction	$P \wedge Q$	both P and Q hold
Disjunction	$P \vee Q$	either P or Q holds
Implication	$P \Rightarrow Q$	if P holds, then Q holds

Examples

P - Bob attends **MATH** course,

Q - Mary is happy

Predicate	
$\neg P$	Bob does not attend MATH course
$P \wedge Q$	Bob attends MATH course and Mary is happy
$P \vee Q$	Bob attends MATH course or Mary is happy
$P \Rightarrow Q$	If Bob attends MATH course, then Mary is happy

Quantified Predicates

We can quantify over a variable of a predicate universally or existentially:

Name	Predicate	Definition
Universal Quantification	$\forall x \cdot P$	P holds for all x
Existential Quantification	$\exists x \cdot P$	P holds for some x



Quantified Predicates

In the predicate $\forall x \cdot P$ the quantification is over all possible values in the type of the variable x .

Typically we constrain the range of values using **implication**.

Examples:

- $\forall x \cdot x > 5 \Rightarrow x > 3$
- $\forall st \cdot st \in registered \Rightarrow st \in STUDENTS$



Quantified Predicates

In the case of **existential quantification** we typically constraint the range of values using **conjunction**.

Example:

- we could specify that integer z has a positive square root as follows:

$$\exists y. y \geq 0 \wedge y^2 = z$$

- $\exists st \cdot st \in STUDENTS \wedge st \notin registered$



Examples

$DATABASE = \{Bill, Ben, Anna, Alice\}, MATH = \{Alice, Ben\}$

TRUE or FALSE?

$Alice \in DATABASE$

$Anna \in MATH$

$\forall x \cdot x \in DATABASE \Rightarrow x \in MATH$

$\exists x. x \in MATH \wedge x \in DATABASE$

$\forall x \cdot x \in MATH \Rightarrow x \in DATABASE$



Examples

$DATABASE = \{Bill, Ben, Anna, Alice\}$, $MATH = \{Alice, Ben\}$

$Alice \in DATABASE$ **TRUE**

$Anna \in MATH$ **FALSE**

$\forall x. x \in DATABASE \Rightarrow x \in MATH$ **FALSE**

$\exists x. x \in MATH \wedge x \in DATABASE$ **TRUE**

$\forall x. x \in MATH \Rightarrow x \in DATABASE$ **TRUE**



Free and bound variables

Variables play two different roles in predicate logic:

- A variable that is universally or existentially quantified in a predicate is said to be a **bound** variable.
- A variable referenced in a predicate that is not bound variable is called a **free** variable.
- Example

$$\exists y. y \geq 0 \wedge y^2 = z$$

y is bound while z is free.

This is a property of y and may be true or false depending on what z is.

The role of y is to bind the quantifier \exists and the formula together.

Predicates on Sets

Predicates on sets can be defined in terms of the logical operators as follows:

Name	Predicate	Definition
Subset	$S \subseteq T$	$\forall x \cdot x \in S \Rightarrow x \in T$
Set equality	$S = T$	$S \subseteq T \wedge T \subseteq S$



Duality of universal and existential quantification

$$\neg \forall x \cdot (x \in S \Rightarrow T) = \exists x \cdot (x \in S \wedge \neg T)$$

$$\neg \exists x \cdot (x \in S \wedge T) = \forall x \cdot (x \in S \Rightarrow \neg T)$$

Defining set operators with logic

Name	Predicate	Definition
Negation	$x \notin S$	$\neg(x \in S)$
Union	$x \in S \cup T$	$x \in S \vee x \in T$
Intersection	$x \in S \cap T$	$x \in S \wedge x \in T$
Difference	$x \in S \setminus T$	$x \in S \wedge x \notin T$
Subset	$S \subseteq T$	$\forall x \cdot x \in S \Rightarrow x \in T$
Power set	$x \in \mathbb{P}(T)$	$x \subseteq T$
Empty set	$x \in \emptyset$	FALSE
Membership	$x \in \{a, \dots, b\}$	$x=a \vee \dots \vee x=b$



Predicates in Event-B

- The invariants of an Event-B model and the guards of an event are formulated as predicates.
- The proof obligations generated by Rodin are also predicates.
- A predicate is simply an expression, the value of which is either true or false.



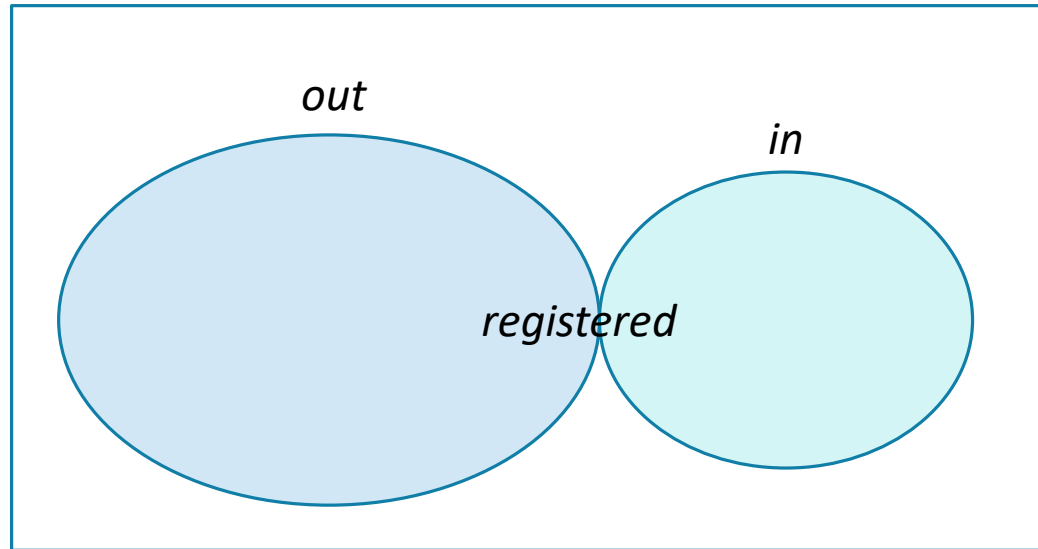
Example: access control to a building

A system for controlling access to a university building

- An university has some fixed number of students.
- Students can be inside or outside the university building.
- The system should allow a new student to be registered in order to get the access to the university building.
- To deny the access to the building for a student the system should support deregistration.
- The system should allow only registered students to enter the university building.

Example: access control to a building

A system for controlling access to a university building



Model context

CONTEXT BuildingAccess_c0

SETS *STUDENTS* //

CONSTANTS *max_capacity* // max capacity of the building is defined as a model constant
(we will need it later, could be skipped for this example)

AXIOMS

axm1: finite(*STUDENTS*)

axm2: max_capacity $\in \mathbb{N}$

axm3: max_capacity > 0

END



Model machine

MACHINE BuildingAccess_m0

SEES BuildingAccess_c0

VARIABLES *registered in out*

*//The machine state is represented by three variables, *registered*, *in*, *out*.*

INVARIANTS

inv1: $registered \subseteq STUDENTS$ // registered students are of type STUDENTS

inv2: $registered = in \cup out$ // registered students are either inside or outside the university building

inv3: $in \cap out = \emptyset$ // no student is both inside and outside the university building

EVENTS ...

EVENTS

INITIALISATION \triangleq

then

act1: *registered, in, out* := $\emptyset, \emptyset, \emptyset$ // initially all the variables are empty

end

ENTER \triangleq // a student entering the building

any *st*

where

grd1: *st* \in *registered* // student must be registered

grd2: *st* \in *out* // student must be outside

then

act1: *in* := *in* \cup {*st*} // add to in

act2: *out* := *out* \setminus {*st*} // remove from out

end

Redundant guard since every student from out is registered

EXIT \triangleq // a student leaves the building

any *st*
where

grd1: $st \in registered$ // a student must be registered

grd2: $st \in in$ // a student must be inside

then

act1: $in := in \setminus \{st\}$ // remove *st* from *in*

end

REGISTER \triangleq // registration a new student

any *st*
where

grd1: $st \in STUDENTS$ // a new student

grd2: $st \notin registered$ // ... that is not in the set registered yet

then

act1: $registered := registered \cup \{st\}$ // add *st* to registered

act2: $out := out \cup \{st\}$ // add *st* to out

end

Redundant guard since every student from out is registered

```
DEREGISTER1  $\triangleq$            // de-register a student
any st
where

    grd1:  $st \in registered$  // a student must be registered
then
    act1:  $registered := registered \setminus \{st\}$  // remove st from registered
    act2:  $in := in \setminus \{st\}$                 // remove st from in
    act3:  $out := out \setminus \{st\}$               // remove st from out
end

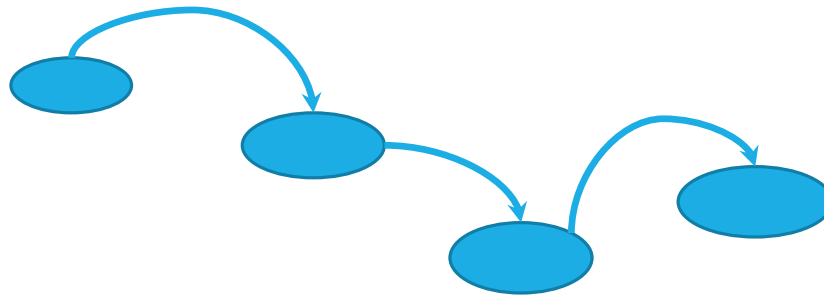
DEREGISTER2  $\triangleq$            // de-register a student while he/she is outside the building
any st
where

    grd1:  $st \in out$            // a new student
then
    act1:  $registered := registered \setminus \{st\}$  // remove st from registered
    act2:  $out := out \setminus \{st\}$                 // remove st from out
end

END
```

Machine behaviour and nondeterminism

- The behaviour of an Event-B machine is defined as a **transition system** that moves from one state to another through execution of events.



- The states of a machine are represented by the different configurations of values for the variables:
 - In our example, the state defined by the variables *registered*, *in*, *out*
 - *NOTE: the order in which events are presented in the specification does not matter. The execution order is defined **only** by the guards*

Machine behaviour and nondeterminism

- In any state that a machine can reach, an enabled event is chosen to be executed to define the next transition.
- If several events are enabled in a state, then the choice of which event occurs is nondeterministic.
- Also, if an event is enabled for several different parameter values, the choice of value for the parameters is nondeterministic – the choice just needs to satisfy the event guards.
 - For example, in the **REGISTER** event, the choice of value for parameter *st* is nondeterministic, with the choice of value being constrained by the guards of the event to ensure that it is a fresh value.
- Treating the choice of event and parameter values as nondeterministic is an abstraction of different ways in which the choice might be made in an implementation of the model.



Relations between sets

- Relation between sets is an important mathematical structure which is commonly used in expressing specifications.
- Relations allow us to express complicated interconnections and relationships between entities formally.

Ordered pairs

- An **ordered pair** is an element consisting of two parts:

a *first* part and *second* part

- An ordered pair with first part x and second part y is written as:

$$x \mapsto y$$

- Examples:

- $(apple \mapsto red)$
- $(Databases \mapsto fall)$
- $(115A \mapsto 30)$
- $(Smith \mapsto 0123)$

• Relationship between an element and a set: is the element a *member* of the set or not?
• For element x and set S , we express the membership relation as follows:
 $x \in S$ (x is a member of S)
where E is a predicate over sets and elements
• *Set membership* is a boolean property relating an element and a set, i.e., either x is in S or is not in S .
• This means that there is no concept of an element occurring more than once in a set, e.g.,
• $\{a, a, b, c\} = \{a, b, c\}$
• $\{b, b\} = \{b\}$
• Conversely, the element is not a member of the set: $x \notin S$

Cartesian product

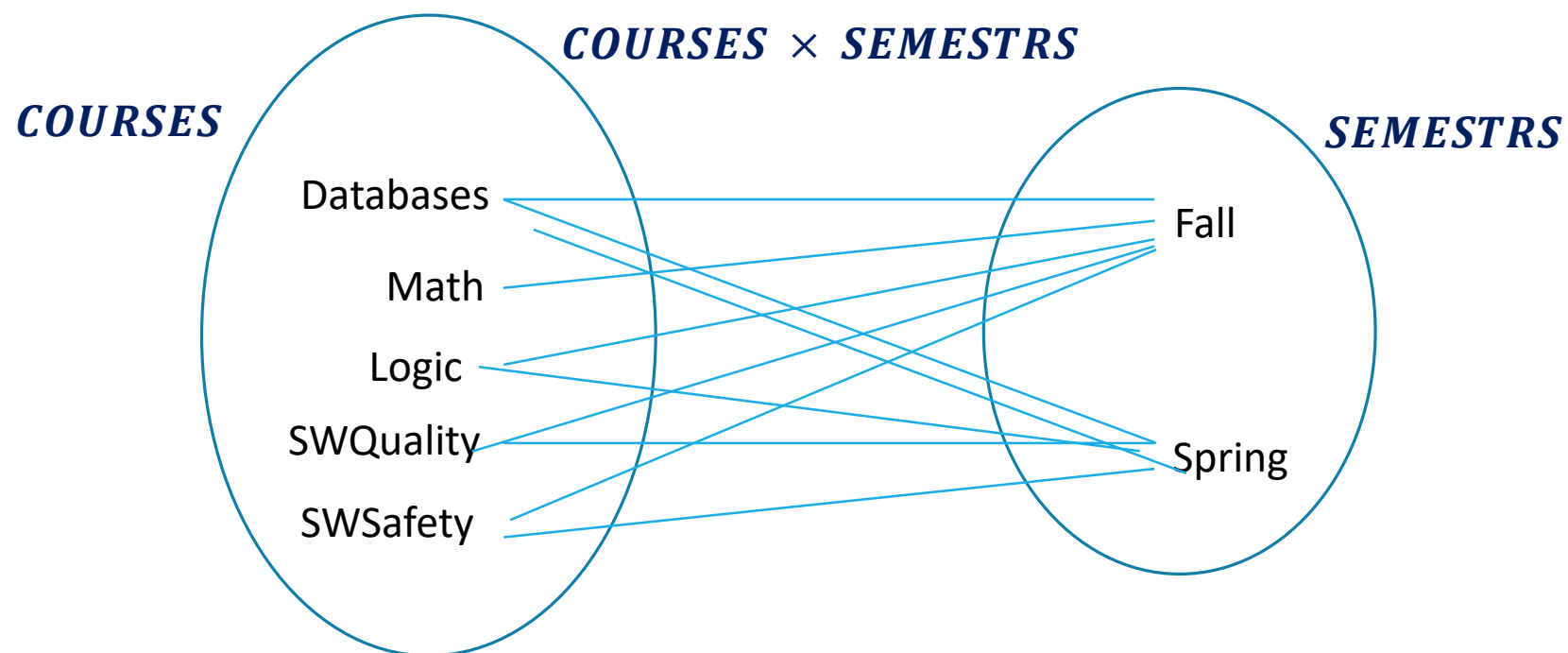
- The **Cartesian product** of two sets is
the **set** of pairs whose first part is in **S** and second part is in **T**
- The Cartesian product of **S** with **T** is written: **$S \times T$**

Cartesian product: example

Lets consider two sets: ***COURSES*** and ***SEMESTERS***



Cartesian product: example





Cartesian product: definition and more examples

- Defining Cartesian product:

Predicate	Definition
$x \mapsto y \in S \times T$	$x \in S \wedge y \in T$

- Examples:

- $\mathbb{N} \times \mathbb{N}$ pairs of natural numbers
- $\{1,2,3\} \times \{a,b\} = \{1 \mapsto a, 1 \mapsto b, 2 \mapsto a, 2 \mapsto b, 3 \mapsto a, 3 \mapsto b\}$
- $\{Anna, Bill, Jack\} \times \emptyset = \emptyset$
- $\{\{1\}, \{1,2\}\} \times \{a,b\} = \{\{1\} \mapsto a, \{1\} \mapsto b, \{1,2\} \mapsto a, \{1,2\} \mapsto b\}$
- $\mathbf{card}(\{yes, no\} \times \{a,b\}) = \mathbf{card}(\{yes \mapsto a, yes \mapsto b, no \mapsto a, no \mapsto b\}) = 4$

Cartesian product is a type constructor

- $\mathcal{S} \times \mathcal{T}$ is a new type constructed from types \mathcal{S} and \mathcal{T} .
- Cartesian product is the type constructor for ordered pairs.
- Given $x \in \mathcal{S}$ and $y \in \mathcal{T}$ we have $x \mapsto y \in \mathcal{S} \times \mathcal{T}$
- Examples:
 - $4 \mapsto 7 \in \mathbb{Z} \times \mathbb{Z}$
 - $\{2, 3\} \mapsto 4 \in \mathbb{P}(\mathbb{Z}) \times \mathbb{Z}$
 - $\{2 \mapsto 1, 3 \mapsto 3, 4 \mapsto 5\} \in \mathbb{P}(\mathbb{Z} \times \mathbb{Z})$



Sets of order pairs

A simple database can be modelled as a set of ordered pairs:

$$\mathbf{studentCourses} = \{Anna \mapsto Logic, Ben \mapsto SWQuality, Jack \mapsto SWQuality, Irum \mapsto$$



Relations

- A **relation** R between sets S and T expresses a relationship between elements in S and elements in T :
 - A relation is captured simply as a set of ordered pairs $(s \mapsto t)$ with $s \in S$ and $t \in T$.
- A relation is a common modelling structure so Event-B has a special notation for it:

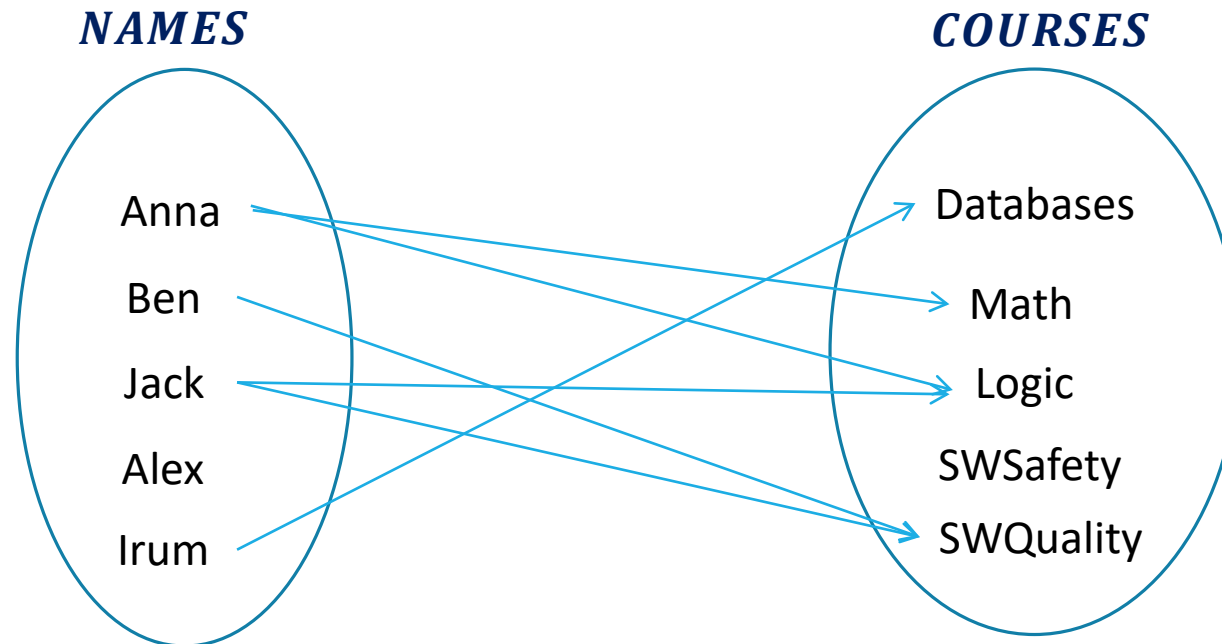
$$S \leftrightarrow T = \mathbb{P}(S \times T)$$

- We can write then

$$\mathbf{studentCourses} = \{Anna \mapsto Logic, Ben \mapsto SWQuality, Jack \mapsto SWQuality, Irum \mapsto$$

Domain and range

studentCourses = {*Anna* \mapsto *Logic*, *Ben* \mapsto *SWQuality*, *Jack* \mapsto *SWQuality*, *Irum* \mapsto



NAMES = {*Anna*, *Ben*, *Jack*, *Alex*, *Irum*}

COURSES = {*Databases*, *Math*, *Logic*, *SWSafety*, *SWQuality*}

Domain

- The **domain** of a relation R is the **set** of first parts of all the pairs in R , written $dom(R)$

Predicate	Definition
$x \in dom(R)$	$\exists y. x \mapsto y \in R$

$studentCourses = \{Anna \mapsto Logic, Ben \mapsto SWQuality, Jack \mapsto SWQuality, Irum \mapsto$

Range

- The **range** of a relation R is the **set** of second parts of all the pairs in R , written $\mathbf{ran}(R)$

Predicate	Definition
$y \in \mathbf{ran}(R)$	$\exists x . x \mapsto y \in R$

$\mathbf{studentCourses} = \{Anna \mapsto Logic, Ben \mapsto SWQuality, Jack \mapsto SWQuality, Irum \mapsto$

Relational image definition

- Assume $R \in S \leftrightarrow T$ and $A \subseteq S$
- The **relational image** of set A under relation R is written $R[A]$

Predicate	Definition
$y \in R[A]$	$\exists x. x \in A \wedge x \mapsto y \in R$

Relational image examples

- ***studentCourses*** = {*Anna* \mapsto *Logic*, *Ben* \mapsto *SWQuality*, *Jack* \mapsto *SWQuality*, *Irum* \mapsto

Partial functions

- Special kind of relation: each domain element has at most one range element associated with it.
- To declare f as a partial function:

$$f \in X \dashrightarrow Y$$

- This says that f is a **many-to-one** relation.
- It is said to be partial because there may be values in the set X that are not in the domain of f
- Each domain element is mapped to one range element:

$$x \in \text{dom}(f) \implies \text{card}(f[\{x\}]) = 1$$

- More often is formalised as a uniqueness constraint

$$x \mapsto y_1 \in f \wedge x \mapsto y_2 \in f \implies y_1 = y_2$$

Function Application

We can use functional application for partial functions

- If $x \in \text{dom}(f)$, then we write $f(x)$ for the unique range element associated with x in f .
- if $x \notin \text{dom}(f)$, then $f(x)$ is undefined.
- if $\text{card}(f[\{x\}]) > 1$, then $f(x)$ is undefined.

Name	Expression	Meaning	Well-definedness
Function application	$f(x)$	$f(x) = y \Leftrightarrow x \mapsto y \in f$	$f \in X \rightarrow Y$ $\wedge x \in \text{dom}(f)$



Examples

$NAMES = \{Anna, Ben, Jack, Alex, Irum\}$, $MNUMBERS = \{0123, 1230, 2301, 3012\}$

studentNumber1 = $\{Anna \mapsto 0123, Ben \mapsto 1230, Irum \mapsto 3012\}$

- ***studentNumber1*** $\in NAMES \rightarrow MNUMBERS$

studentNumber1(Ben)=1230

studentNumber1(Jack) is undefined

Domain Restriction

- Given relation $R \in S \leftrightarrow T$ and $A \subseteq S$, the **domain restriction** of R by A is written
 $A \triangleleft R$
- Restrict relation R so it only contains pairs whose first part is in the set A (keep only those pairs whose first element is in A)
- Example:

$\text{fruitColor} = \{green \mapsto grape, yellow \mapsto banana, red \mapsto apple\}$

$\{red, pink\} \triangleleft \text{fruitColor} = \{red \mapsto apple\}$

Domain Subtraction

- Given $R \in S \leftrightarrow T$ and $A \subseteq S$ the domain subtraction of R by A is written

$$A \triangleleft R$$

- Remove those pairs from relation R whose first part is in the set A (keep only those pairs whose first element NOT in A)

- Example:

$\text{fruitColor} = \{green \mapsto grape, yellow \mapsto banana, red \mapsto apple\}$

$\{red, pink\} \triangleleft \text{fruitColor} = \{green \mapsto grape, yellow \mapsto banana\}$

Range Restriction

- Given $R \in S \leftrightarrow T$ and $A \subseteq T$ the range restriction of R by A is written

$$R \triangleright A$$

- Restrict relation R so the it only contains pairs whose second part is in the set A (keep only those pairs whose second element is in A)

- Example:

$$\begin{aligned} \textit{fruitColor} &= \{ \textit{green} \mapsto \textit{grape}, \textit{yellow} \mapsto \textit{banana}, \textit{red} \mapsto \textit{apple} \} \\ \textit{fruitColor} \triangleright \{ \textit{grape}, \textit{pear} \} &= \{ \textit{green} \mapsto \textit{grape} \} \end{aligned}$$

Range Subtraction

- Given $R \in S \leftrightarrow T$ and $A \subseteq T$ the range subtraction of R by A is written

$$R \triangleright A$$

- Remove those pairs from relation R whose second part is in the set A (keep only those pairs whose second element NOT in A)

- Example:

$$\mathit{fruitColor} = \{green \mapsto grape, yellow \mapsto banana, red \mapsto apple\}$$

$$\mathit{fruitColor} \triangleright \{grape, banana\} = \{red \mapsto apple\}$$

Domain and range, restriction and subtraction: summary

Assume $R \in S \leftrightarrow T$ and $A \subseteq S, B \subseteq T$

Predicate	Definition	Name
$x \mapsto y \in A \triangleleft R$	$x \mapsto y \in R \wedge x \in A$	Domain restriction
$x \mapsto y \in A \triangleleft R$	$x \mapsto y \in R \wedge x \notin A$	Domain subtraction
$x \mapsto y \in R \triangleright B$	$x \mapsto y \in R \wedge y \in B$	Range restriction
$x \mapsto y \in R \triangleright B$	$x \mapsto y \in R \wedge y \notin B$	Range subtraction

Function Overriding

- Override the function f by the function g :

$$f \Leftarrow g$$

- Function f is updated according to g (Override: replace existing mapping with new ones)
- f and g must be partial functions of the same type

Function overriding definition

- Definition in terms of function override and set union

$$f \triangleleft \{a \mapsto b\} = (\{a\} \triangleleft f) \cup \{a \mapsto b\}$$

$$f \triangleleft g = (\text{dom}(g) \triangleleft f) \cup g$$

- Examples:

$$\mathbf{studentNumber} = \{Anna \mapsto 0123, Ben \mapsto 1230, Jack \mapsto 2301, Irum \mapsto 3012\},$$

$$g = \{Ben \mapsto 5555\}$$

$$\mathbf{studentNumber} \triangleleft g = \{Anna \mapsto 0123, Ben \mapsto 5555, Jack \mapsto 2301, Irum \mapsto 3012\}$$

$$g1 = \{Ben \mapsto 5555, Anna \mapsto 1111\}$$

$$\mathbf{studentNumber} \triangleleft g1 = \{Anna \mapsto 1111, Ben \mapsto 5555, Jack \mapsto 2301, Irum \mapsto 3012\}$$



Relation and function

Any operation applicable to a relation or a set is also applicable to a function

- domain and range of a function, range restriction, etc.

If f is a function , then $f(x)$ is the result of function f for the argument x .

Total Functions

- A **total function** is a special kind of partial function. Declaration f as a total function

$$f \in X \rightarrow Y$$

- This means that f is well-defined for every element in X , i.e., $f \in X \rightarrow Y$ is shorthand for

$$f \in X \twoheadrightarrow Y \wedge \text{dom}(f) = X$$



Total injective function

Function called total **injective** (or **1-1**), if for every element y from its range there exists only one element x in the domain and $\mathit{dom}(f) = X$. Declaration f

$$f \in X \rightarrow Y$$

- Example:

$$\mathit{username} \in \mathit{USERS} \rightarrow \mathit{UNAMES}$$

Every user in a system has one unique user name.



Total surjective function

Function called **surjective**, denoted as

$$f \in X \twoheadrightarrow Y$$

if its range is the whole target and $\mathbf{ran}(f) = Y$.

- **Example**

f – “attends school”

$$f \in STUDENTS \twoheadrightarrow SCHOOLS$$

- No school without students (full set $SCHOOLS$ is covered).



Bijjective function

Function is **bijjective**, if it is total, injective and surjective:

$$f \in X \twoheadrightarrow Y$$

- Example

“Married to” – is **bijjective** function,

X - set of “married man”

Y - set of “married woman”

Wrap-up

We refreshed the basics of set theory and predicate logic.

Sometimes even such simple constructs as sets can be used to specify some interesting behaviour but we need to define the relations between the variables in the invariant

Relations and functions allow us specify more complex data structures, i.e., automatically connect sets by defining a specific relation or functions between them

In the specifications, the properties of these mathematical structures would define certain constraints, e.g., if you defined a function then you cannot add to it a pair with the same first element as in some existing pair but different second element because it breaks the definition of what function is. Similarly to other properties (injective, surjective, bijective)

More examples to follow