DD2460 Software safety and security. Lecture 4

ON THE USE OF SET THEORY, FUNCTIONS AND RELATIONS IN EVENT-B MODELLING

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,…}
  - {red, green, black}
  - {yes, no}
  - {wait, start, process, stop}
  - But **not:** {1, 2, 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 $\boldsymbol{x}$ and set $\boldsymbol{S}$, we express the membership relation as follows

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

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 that 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: $\boldsymbol{x} \notin \boldsymbol{S}$

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 $'\}'$:
  - $\mathbf{LectureHall} = \{1A, 1B, 1C, 1D\}$
  - $\boldsymbol{SEMESTRS} = \{spring, fall\}$
- Some sets have predefined names:
  - $\mathbb{N} - the\ set\ of\ natural\ numbers\ \{0, 1, 2, 3, \dots\}$
  - $\mathbb{Z}\ - the\ set\ of\ integers\ \{\dots - 2, -1,\ 0, 1, 2, \dots\}$
- The empty set contains no elements at all. It is the **smallest** possible set.

$$\emptyset \quad or \quad \{\}$$

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:

$$\{\, \underbrace{\boldsymbol{x}}_{\text{A variable}} \mid \underbrace{\boldsymbol{x} \in \boldsymbol{T}}_{\text{ranging over …}} \ \wedge \ \underbrace{\boldsymbol{P}(\boldsymbol{x})}_{\text{condition}} \}$$

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

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

Examples on set comprehension

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

More examples on set comprehension

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

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

Answer: $\{1, 16, 27, 64\}$

Subset and equality relations for sets

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

$$\boldsymbol{S} \subseteq \boldsymbol{T}$$

- For example:
  - $\{3, 7\} \subseteq \{1, 2, 3, 5, 7, 9\};$
  - $\{apple, pear\} \subseteq \{apple, banana, pear, grape\}$
  - $\{Jones, White, Jones\} \subseteq \{White, Smith, Jones, Jakson\}$

- A set $\boldsymbol{S}$ is said to be equal to set $\boldsymbol{T}$ when $\boldsymbol{S} \subseteq \boldsymbol{T}$ and $\boldsymbol{T} \subseteq \boldsymbol{S}$

$$\boldsymbol{S} = \boldsymbol{T}$$

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 that 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 my occur multiple times.

Operations on sets (set operators)

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

$$\boldsymbol{S} \cup \boldsymbol{T}$$

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

$$\boldsymbol{S} \cap \boldsymbol{T}$$

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

$$\boldsymbol{S} \setminus \boldsymbol{T}$$

Examples on Set Operators

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

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

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

Set axioms and laws

- Basic axioms
  - Set membership: $\forall x \cdot x \in S$
  - Empty set: $\forall x \cdot x \in \emptyset$
- Fundamental laws (can be proven)
  - **Commutative laws:**

    $\boldsymbol{S \cup T = T \cup S}$

    $\boldsymbol{S \cap T = T \cap S}$
  - **Associative laws**:

    $\boldsymbol{(S \cup T) \cup R = S \cup (T \cup R)}$

    $\boldsymbol{(S \cap T) \cap R = S \cap (T \cap R)}$
  - **Distributive laws**:

    $\boldsymbol{S \cap (T \cup R) = (S \cap T) \cup (S \cap R)}$

    $\boldsymbol{S \cup (T \cap R) = (S \cup T) \cap (S \cup R)}$

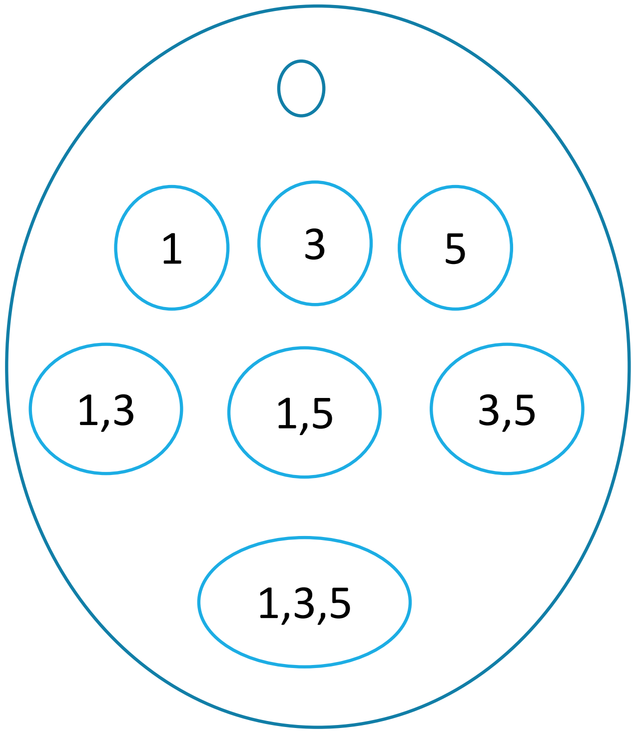
Power sets

- The **power set** of a set $\boldsymbol{S}$ is the set whose elements are all subsets of $\boldsymbol{S}$,

  written $\mathbb{P}(\boldsymbol{S})$

- Example,

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



- $\boldsymbol{S} \in \mathbb{P}(\boldsymbol{T})$ is the same as $\boldsymbol{S} \subseteq \boldsymbol{T}$
- 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 $\boldsymbol{x}$ to be a set of elements of type $\boldsymbol{T}$ we write either

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

More e.g., $\mathrm{math} \subseteq \boldsymbol{COURCES}$ - so type of math is $\mathbb{P}(\boldsymbol{COURCES})$

Cardinality

- The number of elements in a set is called its ***cardinality***
- In Event-B this is written as **card(S)**
- Examples:
  - **card**({1, 2, 3})=3
  - **card**({a, b, c, d})=4
  - **card**({Bill, Anna, Anna, Bill})=2
  - **card**($\mathbb{P}(\{1,3,5\})$)=8
- Cardinality is only defined for finite sets.
  - If S is an infinite set, then ***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, ∅);
  - 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

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

  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 predicate
  - e.g., $\boldsymbol{registered = registered\_spring \cup registered\_fall}$.
- Set membership, e.g., $\mathbf{5} \in \mathbb{N}$
- Subset relations, e.g., $\boldsymbol{S \subseteq T}$
- For integer elements we can write ordering predicates such as $\boldsymbol{x < y}$.

Predicate logic

- **Basic predicates**: $\boldsymbol{x} \in \boldsymbol{S}, \boldsymbol{S} \subseteq \boldsymbol{T}, \boldsymbol{x} \leq \boldsymbol{y}$
- **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:

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

Quantified Predicates

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

Example:

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

$\exists\, y . y \geq 0 \land\ y^2 = z$

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

Examples

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

$Alice \in DATABASE$ TRUE

$Anna \in MATH$ FALSE

$\forall x \cdot x \in DATABASE \Longrightarrow x \in MATH$ FALSE

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

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

$\exists \boldsymbol{y}. \boldsymbol{y} \geq \mathbf{0} \wedge \boldsymbol{y}^{\mathbf{2}} = \boldsymbol{z}$

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	$\boldsymbol{S} \subseteq \boldsymbol{T}$	$\forall \boldsymbol{x} \cdot \boldsymbol{x} \in \boldsymbol{S} \Rightarrow \boldsymbol{x} \in \boldsymbol{T}$
Set equality	$\boldsymbol{S} = \boldsymbol{T}$	$\boldsymbol{S} \subseteq \boldsymbol{T} \ \wedge \ \boldsymbol{T} \subseteq \boldsymbol{S}$

Duality of universal and existential quantification

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

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

Defining set operators with logic

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

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.

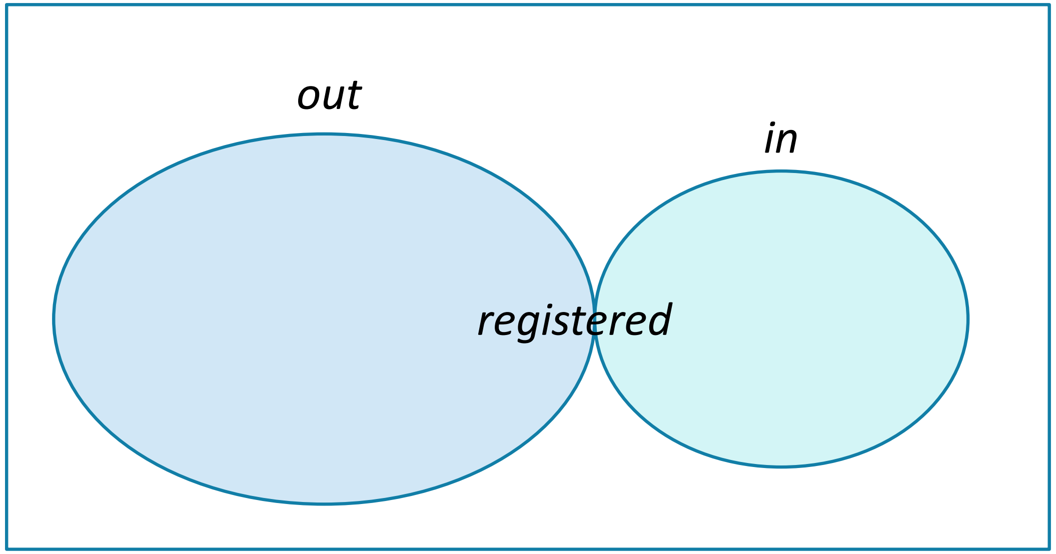
Example: access control to a building

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 in the course lectures)

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 ≙
    then
        act1: registered, in, out := ∅,∅,∅        // initially all the variables are empty
    end

ENTER ≙     //  a student entering the building
    any  st
    where
        grd1: st ∈ registered        // student must be registered
        grd2: st ∈ out               // student must be outside
    then
        act1: in := in ∪ {st}        // add to in
        act2: out := out \ {st}      // remove from out
    end
```

Redundant guard since every student from out is registered

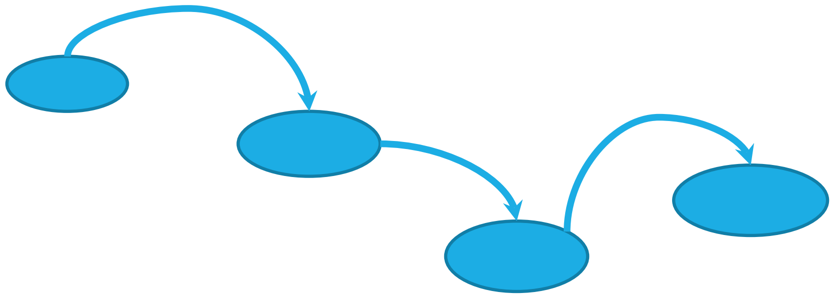
```
EXIT ≙                    // a student leaves the building
    any  st
    where
        grd1: st ∈ registered   // a student must be reg
        grd2: st ∈ in           // a student must be inside
    then
        act1: in := in \ {st}       // remove st from in
        act2: out := out ∪ {st} // remove st from in

    end
REGISTER ≙            // registration a new student
    any  st
    where
        grd1: st ∈ STUDENTS   // a new student
        grd2: st ∉ registered   // … that is not in the set registered yet
    then
        act1: registered := registered ∪ {st}        // add st to registered
        act2: out := out  ∪ {st}                      // add st to out
    end
```

Redundant guard since every student from out is registered

```
DEREGISTER1 ≙          // de-register a student
    any  st
    where
        grd1: st ∈ registered   // a student must be registered
    then
        act1: registered := registered \ {st}  // remove st from registered
        act2: in := in \ {st}                  // remove st from in
        act3: out := out \ {st}                // remove st from out
    end
DEREGISTER2 ≙          // de-register a student while he/she is outside the building
    any  st
    where
        grd1: st ∈ out       // a new student
    then
        act1: registered := registered \ {st}  // remove st from registered
        act2: out := out  \ {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*

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 entitites *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 $\boldsymbol{x}$ and second part $\boldsymbol{y}$ is written as:

$$\boldsymbol{x} \mapsto \boldsymbol{y}$$

- Examples:
  - $(\boldsymbol{apple} \mapsto \boldsymbol{red})$
  - $(\boldsymbol{Databases} \mapsto \boldsymbol{fall})$
  - $(\boldsymbol{115A} \mapsto \boldsymbol{30})$
  - $(\boldsymbol{Smith} \mapsto \boldsymbol{0123})$

Cartesian product

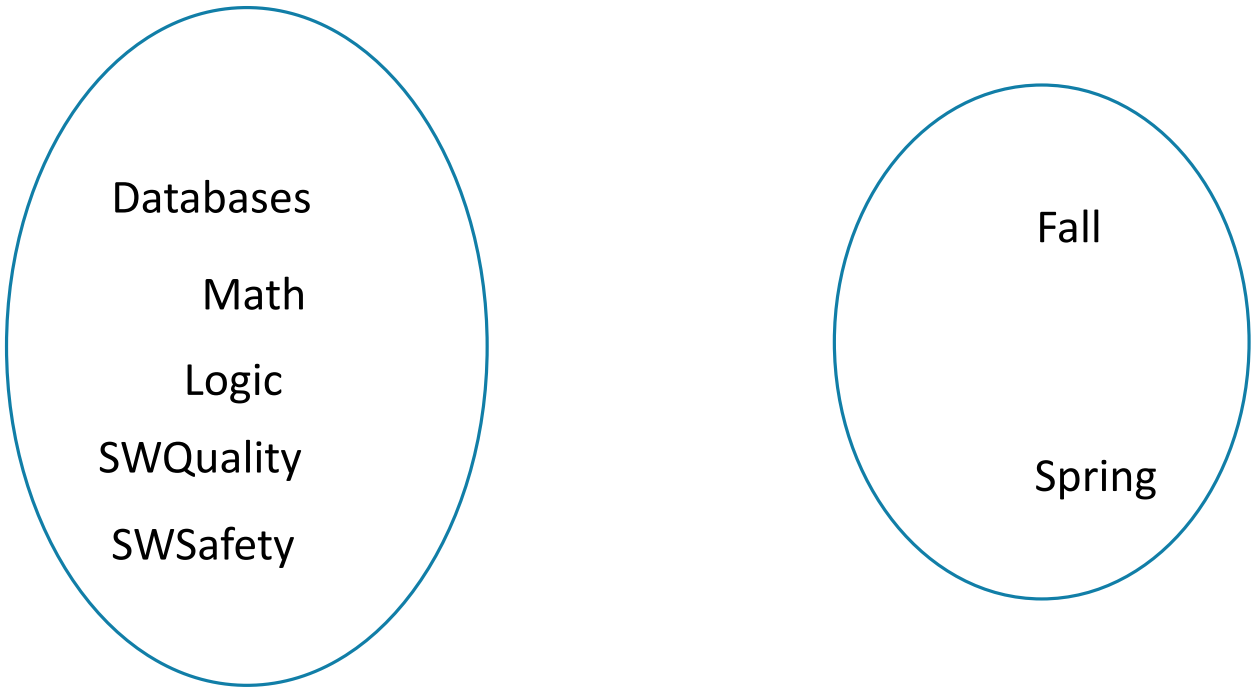
- The **Cartesian product** of two sets is

  the **set** of pairs whose first part is in $\boldsymbol{S}$ and second part is in $\boldsymbol{T}$

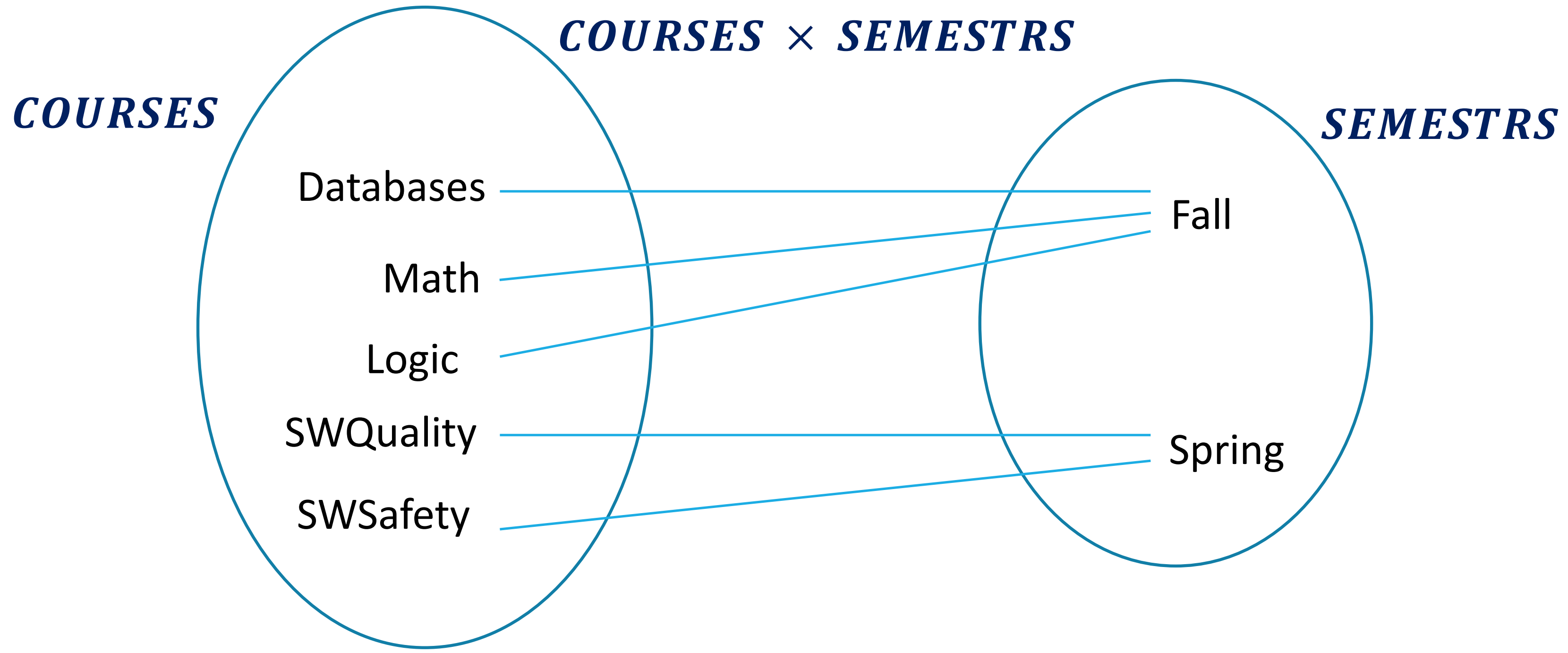
- The Cartesian product of $\boldsymbol{S}$ with $\boldsymbol{T}$ is written: $\boldsymbol{S} \times \boldsymbol{T}$

Cartesian product: example

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



Cartesian product: example



Cartesian product: definition and more examples

- Defining Cartesian product:

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

- 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

- $\boldsymbol{S} \times \boldsymbol{T}$ is a new type constructed from types $\boldsymbol{S}$ and $\boldsymbol{T}$.
- Cartesian product is the type constructor for ordered pairs.
- Given $\boldsymbol{x} \in \boldsymbol{S}$ and $\boldsymbol{y} \in \boldsymbol{T}$ we have $\boldsymbol{x} \mapsto \boldsymbol{y} \in \boldsymbol{S} \times \boldsymbol{T}$

- 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:

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

Relations

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

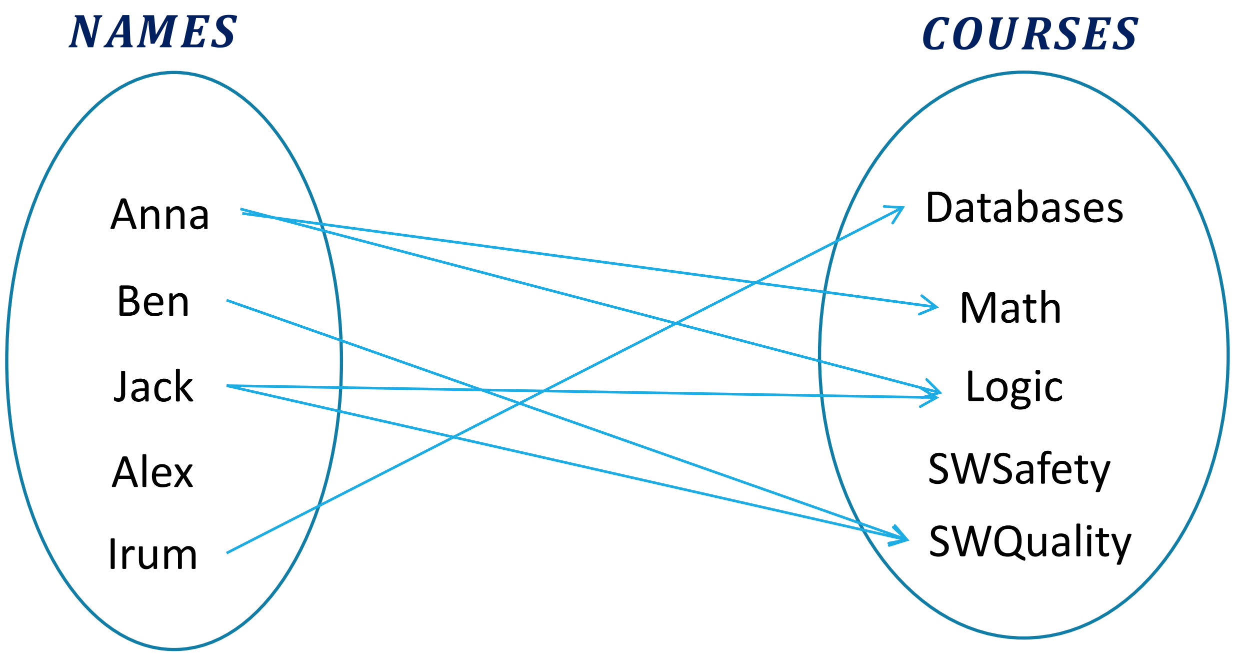
$$\boldsymbol{S} \leftrightarrow \boldsymbol{T} = \mathbb{P}\,(\boldsymbol{S} \times \boldsymbol{T})$$

- We can write then

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

Domain and range

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



$\boldsymbol{NAMES} = \{Anna, Ben, Jack, Alex, Irum\}$ $\boldsymbol{COURSES} = \{Databases, Math, Logic, SWSafety, SWQuality\}$

Domain

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

Predicate	Definition
$\boldsymbol{x} \in \boldsymbol{dom}(\boldsymbol{R})$	$\exists\, \boldsymbol{y}.\ \boldsymbol{x} \mapsto \boldsymbol{y}\ \in \boldsymbol{R}$

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

Range

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

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

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

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
$y \in R[A]$	$\exists\, x.\ x \in A \land x \mapsto y \in R$

Relational image examples

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

Partial functions

- Special kind of relation: each domain element has <u>at most one</u> range element associated with it.
- To declare $\boldsymbol{f}$ as a partial function:

$$\boldsymbol{f} \in \boldsymbol{X} \pto \boldsymbol{Y}$$

- This says that $\boldsymbol{f}$ is a **many-to-one** relation.
- It is said to be partial because there may be values in the set $\boldsymbol{X}$ that are not in the domain of $\boldsymbol{f}$
- Each domain element is mapped to <u>**one**</u> range element:

$$\boldsymbol{x} \in \boldsymbol{dom}(\boldsymbol{f}) \quad \Longrightarrow \quad \boldsymbol{card}(\boldsymbol{f}[\{\boldsymbol{x}\}]) = \mathbf{1}$$

- More usually formalised as a uniqueness constraint

$$\boldsymbol{x} \mapsto \boldsymbol{y_1} \in \boldsymbol{f} \wedge \boldsymbol{x} \mapsto \boldsymbol{y_2} \in \boldsymbol{f} \quad \Longrightarrow \quad \boldsymbol{y_1}=\boldsymbol{y_2}$$

Function Application

We can use functional application for partial functions

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

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

Examples

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

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

$\boldsymbol{studentNumber2} = \{Anna \mapsto 0123, Ben \mapsto 1230, Jack \mapsto 2301, Jack \mapsto 3012\}$

- $\boldsymbol{studentNumber1} \in NAMES \mathrel{+\!\!\!\rightarrow} MNUMBERS$

$\boldsymbol{studentNumber1}(Ben)=1230$

$\boldsymbol{studentNumber1}(Jack)$ is undefined

- $\boldsymbol{studentNumber2} \notin NAMES \mathrel{+\!\!\!\rightarrow} MNUMBERS$

$\boldsymbol{studentNumber2}(Jack)$ is undefined

Domain Restriction

- Given relation $\boldsymbol{R} \in \boldsymbol{S} \leftrightarrow \boldsymbol{T}$ and $\boldsymbol{A} \subseteq \boldsymbol{S},$ the **domain restriction** of $\boldsymbol{R}$ by $\boldsymbol{A}$ is written

$$\boldsymbol{A} \lhd \boldsymbol{R}$$

- Restrict relation $\boldsymbol{R}$ so it only contains pairs whose <u>first</u> part is in the set $\boldsymbol{A}$ (keep only those pairs whose first element is in A)

- Example:

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

$$\{red, pink\} \lhd \boldsymbol{fruitColor} = \{red \mapsto apple\}$$

Domain Subtraction

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

$$\boldsymbol{A} \lhd \boldsymbol{R}$$

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

- Example:

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

$$\{red, pink\} \lhd \boldsymbol{fruitColor} = \{green \mapsto grape, yellow \mapsto banana\}$$

Range Restriction

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

$$\boldsymbol{R} \rhd \boldsymbol{A}$$

- Restrict relation R so the it only contains pairs whose <u>second</u> part is in the set $\boldsymbol{A}$ (keep only those pairs whose second element is in $\boldsymbol{A}$)

- Example:

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

$$\boldsymbol{fruitColor} \rhd \{grape, pear\} = \{green \mapsto grape\}$$

Range Subtraction

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

$$\boldsymbol{R} \rhd\!\!\!- \boldsymbol{A}$$

- Remove those pairs from relation $\boldsymbol{R}$ whose <u>second</u> part is in the set $\boldsymbol{A}$ (keep only those pairs whose second element NOT in $\boldsymbol{A}$)

- Example:

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

$$\boldsymbol{fruitColor} \rhd\!\!\!- \{grape, banana\} = \{red \mapsto apple\}$$

Domain and range, restriction and subtraction: summary

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

Predicate	Definition	Name
$\boldsymbol{x} \mapsto \boldsymbol{y} \in A \lhd R$	$\boldsymbol{x} \mapsto \boldsymbol{y} \in R \wedge \boldsymbol{x} \in A$	Domain restriction
$\boldsymbol{x} \mapsto \boldsymbol{y} \in A \ntriangleleft R$	$\boldsymbol{x} \mapsto \boldsymbol{y} \in R \wedge \boldsymbol{x} \notin A$	Domain subtraction
$\boldsymbol{x} \mapsto \boldsymbol{y} \in R \rhd B$	$\boldsymbol{x} \mapsto \boldsymbol{y} \in R \wedge \boldsymbol{y} \in B$	Range restriction
$\boldsymbol{x} \mapsto \boldsymbol{y} \in R \ntriangleright B$	$\boldsymbol{x} \mapsto \boldsymbol{y} \in R \wedge y \notin B$	Range subtraction

Function Overriding

- Override the function $\boldsymbol{f}$ by the function $\boldsymbol{g}$:

$$\boldsymbol{f} \oplus \boldsymbol{g}$$

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

Function overriding definition

- Definition in terms of function override and set union

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

$$f \oplus g = (dom(g) \lhd f) \cup g$$

- Examples:

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

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

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

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

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

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 $\boldsymbol{f}$ is a function , then $\boldsymbol{f}(\boldsymbol{x})$ is the result of function $\boldsymbol{f}$ for the argument x.

Total Functions

- A **total function** is a special kind of partial function. Declaration f as a total function
$$f \in X \longrightarrow Y$$
- This means that f is well-defined for every element in X, i.e., $f \in X \longrightarrow Y$ is shorthand for
$$f \in X \nrightarrow Y \wedge dom(f) = X$$

Total injective function

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

$$\boldsymbol{f} \in \boldsymbol{X} \rightarrowtail \boldsymbol{Y}$$

- Example:

$\boldsymbol{username} \in \boldsymbol{USERS} \rightarrowtail \boldsymbol{UNAMES}$

Every user in a system has one unique user name.

Total surjective function

Function called **surjective**, denoted as

$$\boldsymbol{f} \in \boldsymbol{X} \twoheadrightarrow \boldsymbol{Y}$$

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

- **Example**

$\boldsymbol{f}$ $-$"$attends\ school$"
$\boldsymbol{f} \in STUDENTS \twoheadrightarrow SCHOOLS$

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

Bijective function

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

$$f \in X \rightarrowtail\!\!\!\!\twoheadrightarrow Y$$

- Example

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

X - set of “married man”

Y - set of “married woman”

Example: printer access for students

The system tracks the permissions that students have with regard to the printers available at the university network.

- A system should support adding a permission for a student in order to get an access to a particular printer and removing a permission.
- A system should support removing a student’s access to all printers at once.
- A system should support giving the combined permissions of any two students to both of them.

Printer access

- Permissions are naturally expressed as a *relation* between students and printers, so the machine makes use of a variable whose type is relation.
- Since the machine will have to keep track of changing permissions, it will make use of a *variable* ***access*** whose type is a *relation* between *STUDENTS* and *PRINTERS*.
- As permissions are added or removed, the variable will be updated to reflect the information.

Printer access: context

```
CONTEXT PrinterAccess_c0
SETS STUDENTS
       PRINTERS
AXIOMS
       axm1: finite(STUDENTS)
       axm2: finite(PRINTERS)
       axm3: STUDENTS≠ ∅
       axm4: PRINTERS≠ ∅
 END
```

Printer access: machine

```
MACHINE PrinterAccess_m0
SEES PrinterAccess_c0
VARIABLES access
INVARIANTS
        inv1: access ∈ STUDENTS ↔ PRINTERS
EVENTS
  INITIALISATION ≙
        begin
              act1: access := ∅
        end
…
```

Model events

ADD $\triangleq$
  any *st pr*
  where
    grd1: *st* ∈ STUDENTS
    grd2: *pr* ∈ PRINTERS
  then
    act1: *access:=access* ∪ $\{st \mapsto pr\}$
  end
BLOCK $\triangleq$
  any *st pr*
  where
    grd1: *st* ∈ STUDENTS
    grd2: *pr* ∈ PRINTERS
    grd3: $st \mapsto pr$ ∈ *access*
  then
    act1: *access:=access* $\setminus\{st \mapsto pr\}$
  end

Model events

```
BAN ≙
    any st
    where
        grd1: st ∈ STUDENTS
    then
        act1: access:={st} ⩤ access
    end
UNIFY ≙
    any st1  st2
    where
        grd1: st1 ∈ STUDENTS
        grd2: st2 ∈ STUDENTS
    then
        act1: access:= access ∪ ({st1} × access[{st2}]) ∪ ({st2} ×
access[{st1}])
    end
END
```

Printer access rules

- Assume that we want to restrict the number of printers that a student can have access to.

For example, a student can use no more than 3 printers.

We have to reflect this new functionality into our model.

Model events: modification of ADD event

ADD $\triangleq$

any *st pr*

where

grd1: $st \in$ STUDENTS

grd2: $pr \in$ PRINTERS

grd3: **???** **// we have to specify new condition here**

then

act1: $access := access \cup \{st \mapsto pr\}$

end

Model events: modification of ADD event

```
ADD ≙
    any st pr
    where
        grd1: st ∈ STUDENTS
        grd2: pr ∈ PRINTERS
        grd3: card({st}◁access) < 3        // new guard
    then
        act1: access:=access ∪ {st ↦ pr}
    end
```

// We restrict a domain of ***access*** *relation by a set containing one element student* ***st****, i.e.,* $\{st\} \lhd$ ***access***. As a result of this operation we get a set of pairs, whose the first element is ***st***. Then by ***card*** operator we count a number of such pairs. Thus, we get a number of printers that this particula student ***st*** has access to.

Model events: modification of UNIFY event

Similarly, we have to modify the event **UNIFY.**

However, the new guard here will be rather complex

- *Informally:* we have to check, if, after the Unify operation, two students still will have access to no more than 3 printers.

This means that the following property should be defined as a model invariant (and, consequently preserved during events execution):

$$\forall\, st.\ st \in \boldsymbol{dom}(\boldsymbol{access}) \Rightarrow \boldsymbol{card}(\{st\} \lhd \boldsymbol{access}) \leq 3$$

More examples

- ***Every person is either a student or a lecturer. But no person can be a student and a lecturer at the same time.***

$STUDENTS \subseteq PERSONS, LECTURERS \subseteq PERSONS$

$LECTURERS \cup STUDENTS = PERSONS$

$LECTURERS \cap STUDENTS = \emptyset$

- ***Only lecturer can teach course***

$e.g., \; CourseLecturer \; \in COURSES \leftrightarrow LECTURERS$

More examples

- ***Every* course is given by *at most one* lecturer**

$CourseLecturer \in COURSES \longrightarrow LECTURERS$ // total function

- ***A lecturer has to teach at least one course and at most three courses***

$CourseLecturer \in COURSES \longrightarrow LECTURERS \land \boldsymbol{ran}(CourseLecturer) = LECTURERS$

$\land (\forall\, l.\ \boldsymbol{card}(CourseLecturer \rhd \{l\}) \leq 3))$

Comment on Initialisation event

MACHINE *CoursesRegistration_m0*
SEES *CoursesRegistration_m0*
VARIABLES *access*
INVARIANTS
  inv1: $CourseLecturer \in COURSES \longrightarrow LECTURERS$
  ….
EVENTS
 INITIALISATION ≙
  begin
   act1: $CourseLecturer := \emptyset$ *// wrong! Since* $CourseLecturer$ *defined as a total function*
  end

inv1 invariant should be preserved upon **INITIALISATION** event.

BUT Rodin prover will fail to prove that since upon substitution $CourseLecturer$ by $\emptyset$, it will have to prove that $\emptyset \in COURSES \longrightarrow LECTURERS$. But it is wrong!

Simple example: seat booking system

The system allows a person to make a seat booking. Specifically:

- A system should support booking a seat by only one person;
- A system should support cancelling of a booking.

Modelling seat booking system in Event-B

- In the static part of our Event-B model – context - we will introduce required sets: *SEATS* and *PERSONS* as well as required axioms.
- In the dynamic part of the model – machine – we will define (specify) operations by events **BOOK** and **CANCEL**, correspondingly.
- We introduce a variable ***booked_seats*** whose type is a *partial function* on the sets *SEATS* and *PERSONS.*
- ***booked_seats*** keeps a track on booked seats and persons make their booking.
- Since booking of a seat can be done or cancelled, the variable ***booked_seats*** will be updated by the events **BOOK** or **CANCEL** to reflect this.

Seat booking system

We define a context **BookingSeats_c0** as follows

```
CONTEXT
      BookingSeats_c0
SETS
      PERSONS
      SEATS
AXIOMS
      axm1: finite(SEATS)
      axm2: finite(PERSONS)
      axm3: SEATS≠ ∅
      axm4: PERSONS≠ ∅
END
```

Machine **BookingSeats_m0**

MACHINE **BookingSeats_m0**
SEES **BookingSeats_c0**
VARIABLES
booked_seat
INVARIANTS
inv1: booked_seat $\in$ *SEATS* $⇸$ *PERSONS*
// this variable is defined as a partial function (every seat can be occupied by only one person, but not every seat from the set SEATS is booked yet)
EVENTS
INITIALISATION ≙
then
act1: booked_seat := ∅ // empty set
end
BOOK ≙ //booking a seat
any person seat
where
grd1: person ∈ *PERSONS* // take any person
grd2: seat ∈ *SEATS* // we take any seat ...
grd3: seat ∉ dom(booked_seat) // ... that is free
then
act1: booked_seat := *booked_seat* ∪ {seat ↦ person}
end

CANCEL ≙ // cancelation of booking
any *person seat*
where
grd1: seat ↦ person ∈ booked_seat // any pair from booked_seat
then
act1: booked_seat := booked_seat \ {seat ↦ person}
// delete this pair from *booked_seat*
end
END