DD2460 Software safety and security. Lecture 4

ON THE USE OF SET THEORY, MODELLING WITH SETS

Lecture outline

Basic set theory

Examples of using sets in Event-B modelling

Predicates

Simple Event-B specification: access to university buildings

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.,
  - $\{a, a, b, c\} = \{a, b, c\}$;
  - $\{3, 7\} = \{3, 7, 7\}$
- Elements may themselves be sets, i.e., we can have a set of sets.
- 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:
  - $\mathbb{N} - the\ set\ of\ natural\ numbers\ \{0, 1, 2, 3, \ldots\}$
  - $\mathbb{Z}\ -the\ set\ of\ integers\ \{\ldots-2, -1,\ 0, 1, 2, \ldots\}$
- 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: $\{x \mid x \in \mathbb{N} \wedge x < 10\}$
  - 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: $\{1, 8, 27, 64\}$

Sets in Event-B specifications

Example: we want to define a variable *current_season* which models current season

CONTEXT C

SETS

SEASONS = {SPRING, SUMMER, AUTUMN, WINTER}

CONSTANTS

SPRING

SUMMER

AUTUMN

WINTER

MACHINE M

SEES Context C

VARIABLES

current_season

INVARIANT

current_season ∈ SEASONS

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

- Fundamental laws (can be proven)
  - **Commutative laws:**

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

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

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

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

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

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

Group activity

Challenge 1 (3 min):

Imagine that you need to model a drink dispenser. The basic functionality would be a user comes and selects a drink and the machine dispenses it. (We assume a money-free word for now). Which variable and which type do you need to define?

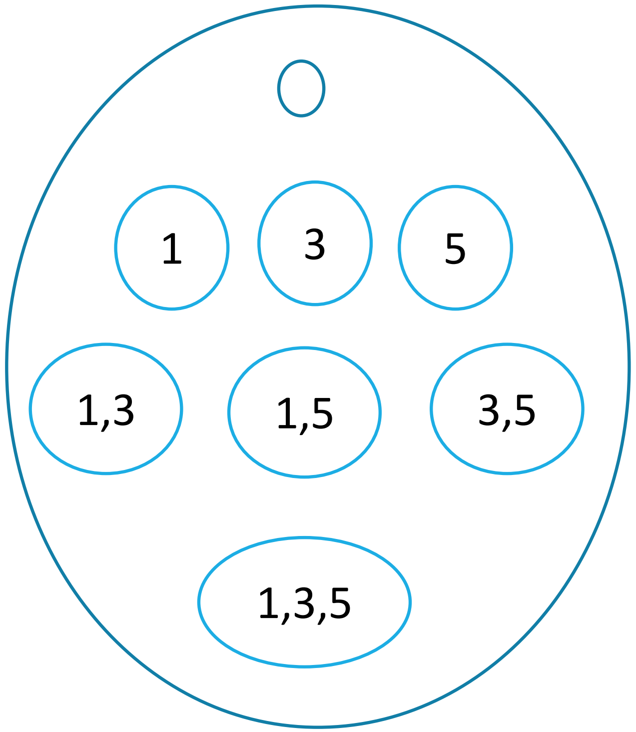
Power sets

- 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 $\mathbb{P}(\boldsymbol{COURCES})$

More on “seasons” example

If we ant to define two variables warm_seasons and cold_seasons:

For Sweden (with some optimism)

warm_seasons = {SPRING,SUMMER}

cold_seasons= {AUTUMN, WINTER}

For Hawaii

warm_seasons = {SPRING,SUMMER, AUTUMN, WINTER}

cold_seasons = $\emptyset$

In both cases *warm_seasons* and *cold_seasons* variable have type $\mathbb{P}$(SEASONS)

Group activity

Challenge 1 (3 min): done

Imagine that you need to model a drink dispenser. The basic functionality would be a user comes and selects a drink and the machine dispenses it. (We assume a money-free word for now). Which variable and which type do you need to define?

Challenge 2 (5 min): Now you need to model a smart drink dispenser. It categorises the drinks into healthy and not-so-healthy. The dispenser first asks the user to make a choice of healthy or unhealthy drink. Then it asks the selection criteria: contains no sugar or no caffeine. Then it shows the corresponding options. Which variables do you need to introduce? What are the types?

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, $\emptyset$);
  - 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

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

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

Predicate logic

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

Name	Predicate	Definitions
Negation	$\neg P$	P does not hold
Conjunction	$P \land Q$	both P and Q hold
Disjunction	$P \lor Q$	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:

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

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

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

$\exists\, x. x \in MATH \land x \in DATABASE$ 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	$x \notin S$	$\neg(x \in S)$
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	$x \in \{a,\ldots, b\}$	$x=a \vee \ldots \vee x=b$

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.

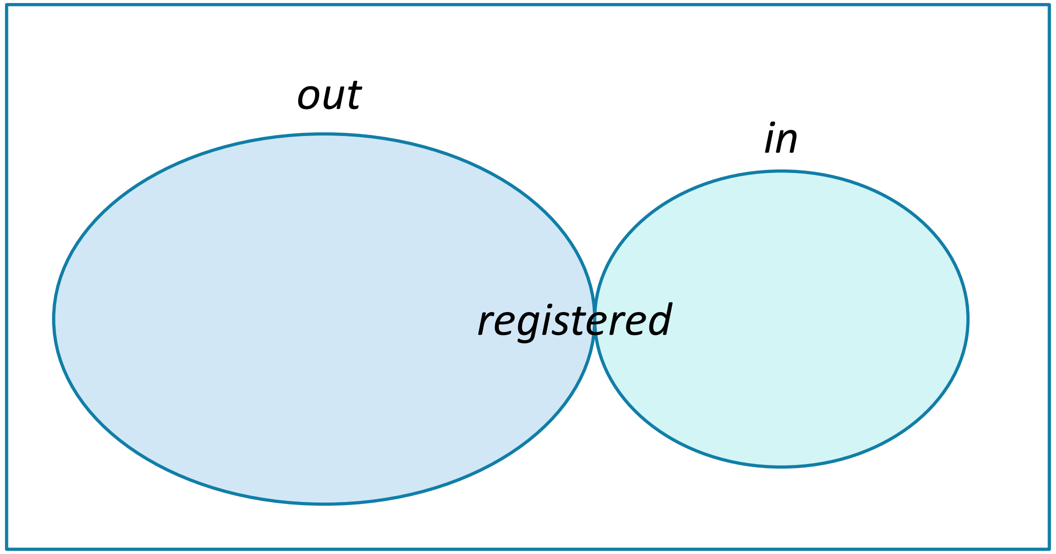
Example: access control to a building

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 in the course lectures)
AXIOMS
      axm1: finite(STUDENTS)
      axm2: max_capacity ∈ 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 $\triangleq$ // a student leaves the building

Redundant guard since every student from *in* is registered

```
    any  st
    where
        grd1: st ∈ registered     // a student must be registered
        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
```

```
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
```

Wrap-up

We have refreshed the knowledge about set theory and predicate logic

Formal specification in Event-B is about using them to abstractly describe behaviour of the system

We rely on properties of different mathematical structures to implicitly state some properties

For example, when we defined a carrier set STUDENTS, we have implicitly defined the constrain for the eventual implementation of the student registration management system: there no two identical students, i.e. even if the some students have identical names they id should be different

Observe that invariant enforces us to be very precise