

FIT3013 Lecture Notes

Revision of Set Theory

John Hurst, 2009

Clayton School of Information Technology
Monash University, Melbourne, Australia

20090311 / Lecture 4

Outline

Objectives

Defining Sets

Operations on Sets

Relations

Functions

Summary

Objectives for this Module

- 1 an understanding of the Fundamentals of the Event-B Method
- 3 A reading knowledge of Event-B specifications
- 9 Skills in using the Event-B notation to develop and prove software specifications
- 12 The ability to write basic Event-B specifications

Objectives for this Module

- 1 an understanding of the Fundamentals of the Event-B Method
- 3 A reading knowledge of Event-B specifications**
- 9 Skills in using the Event-B notation to develop and prove software specifications
- 12 The ability to write basic Event-B specifications

Objectives for this Module

- 1 an understanding of the Fundamentals of the Event-B Method
- 3 A reading knowledge of Event-B specifications
- 9 Skills in using the Event-B notation to develop and prove software specifications
- 12 The ability to write basic Event-B specifications

Objectives for this Module

- 1 an understanding of the Fundamentals of the Event-B Method
- 3 A reading knowledge of Event-B specifications
- 9 Skills in using the Event-B notation to develop and prove software specifications
- 12 The ability to write basic Event-B specifications

Set Theory

- Sets are **unordered collections of elements**
- Elements are usually named with lower case letters
- Sets are usually named with capital letters
- Concept of **set membership**
- Example: *barina* \in *HOLDENS*

Set Theory

- Sets are **unordered collections** of **elements**
- Elements are usually named with lower case letters
- Sets are usually named with capital letters
- Concept of **set membership**
- Example: *barina* \in *HOLDENS*

Set Theory

- Sets are **unordered collections** of **elements**
- Elements are usually named with lower case letters
- Sets are usually named with capital letters
- Concept of **set membership**
- Example: *barina* \in *HOLDENS*

Set Theory

- Sets are **unordered collections** of **elements**
- Elements are usually named with lower case letters
- Sets are usually named with capital letters
- Concept of **set membership**
- Example: *barina* \in *HOLDENS*

Set Theory

- Sets are **unordered collections** of **elements**
- Elements are usually named with lower case letters
- Sets are usually named with capital letters
- Concept of **set membership**
- Example: *barina* \in *HOLDENS*

Defining Sets

- Two basic ways:
 - enumeration
 - comprehension
- Enumeration: $\{barina, astra, commodore\}$
- Comprehension: $\{x \mid x \in S \wedge P\}$
 - S defines the type of x
 - P is a predicate in x , constraining the set
- Example: $\{x \mid x \in \mathbb{N} \wedge x \leq 10\}$
 - read (*the set of natural numbers*) x such that x at most 10
 - note a) typing, and b) constraint

Defining Sets

- Two basic ways:
 - enumeration
 - comprehension
- Enumeration: $\{barina, astra, commodore\}$
- Comprehension: $\{x \mid x \in S \wedge P\}$
 - S defines the type of x
 - P is a predicate in x , constraining the set
- Example: $\{x \mid x \in \mathbb{N} \wedge x \leq 10\}$
 - read (*the set of natural numbers*) x such that x at most 10
 - note a) typing, and b) constraint

Defining Sets

- Two basic ways:
 - enumeration
 - comprehension
- Enumeration: $\{barina, astra, commodore\}$
- Comprehension: $\{x \mid x \in S \wedge P\}$
 - S defines the type of x
 - P is a predicate in x , constraining the set
- Example: $\{x \mid x \in \mathbb{N} \wedge x \leq 10\}$
 - read (*the set of natural numbers*) x such that x at most 10
 - note a) typing, and b) constraint

Defining Sets

- Two basic ways:
 - enumeration
 - comprehension
- Enumeration: $\{\textit{barina, astra, commodore}\}$
- Comprehension: $\{x \mid x \in S \wedge P\}$
 - S defines the type of x
 - P is a predicate in x , constraining the set
- Example: $\{x \mid x \in \mathbb{N} \wedge x \leq 10\}$
 - read (the set of natural numbers) x such that x at most 10
 - note a) typing, and b) constraint

Defining Sets

- Two basic ways:
 - enumeration
 - comprehension
- Enumeration: $\{barina, astra, commodore\}$
- Comprehension: $\{x \mid x \in S \wedge P\}$
 - S defines the type of x
 - P is a predicate in x , constraining the set
- Example: $\{x \mid x \in \mathbb{N} \wedge x \leq 10\}$
 - read (the set of natural numbers) x such that x at most 10
 - note a) typing, and b) constraint

Defining Sets

- Two basic ways:
 - enumeration
 - comprehension
- Enumeration: $\{barina, astra, commodore\}$
- Comprehension: $\{x \mid x \in S \wedge P\}$
 - S defines the type of x
 - P is a predicate in x , constraining the set
- Example: $\{x \mid x \in \mathbb{N} \wedge x \leq 10\}$
 - read (the set of natural numbers) x such that x at most 10
 - note a) typing, and b) constraint

Defining Sets

- Two basic ways:
 - enumeration
 - comprehension
- Enumeration: $\{barina, astra, commodore\}$
- Comprehension: $\{x \mid x \in S \wedge P\}$
 - S defines the type of x
 - P is a predicate in x , constraining the set
- Example: $\{x \mid x \in \mathbb{N} \wedge x \leq 10\}$
 - read (*the set of natural numbers*) x such that x at most 10
 - note a) typing, and b) constraint

Defining Sets

- Two basic ways:
 - enumeration
 - comprehension
- Enumeration: $\{barina, astra, commodore\}$
- Comprehension: $\{x \mid x \in S \wedge P\}$
 - S defines the type of x
 - P is a predicate in x , constraining the set
- Example: $\{x \mid x \in \mathbb{N} \wedge x \leq 10\}$
 - read (*the set of natural numbers*) x such that x at most 10
 - note a) typing, and b) constraint

Defining Sets

- Two basic ways:
 - enumeration
 - comprehension
- Enumeration: $\{barina, astra, commodore\}$
- Comprehension: $\{x \mid x \in S \wedge P\}$
 - S defines the type of x
 - P is a predicate in x , constraining the set
- Example: $\{x \mid x \in \mathbb{N} \wedge x \leq 10\}$
 - read (*the set of natural numbers*) x such that x at most 10
 - note a) typing, and b) constraint

Defining Sets

- Two basic ways:
 - enumeration
 - comprehension
- Enumeration: $\{barina, astra, commodore\}$
- Comprehension: $\{x \mid x \in S \wedge P\}$
 - S defines the type of x
 - P is a predicate in x , constraining the set
- Example: $\{x \mid x \in \mathbb{N} \wedge x \leq 10\}$
 - read (*the set of natural numbers*) x such that x at most 10
 - note a) typing, and b) constraint

New Sets from Old

- Union: $S \cup R$
- Intersection: $S \cap R$
- Powerset: $\mathbb{P}(S)$
- Cartesian Product: $S \times R$

See the B summary for formal definitions of these

New Sets from Old

- Union: $S \cup R$
- **Intersection: $S \cap R$**
- Powerset: $\mathbb{P}(S)$
- Cartesian Product: $S \times R$

See the B summary for formal definitions of these

New Sets from Old

- Union: $S \cup R$
- Intersection: $S \cap R$
- Powerset: $\mathbb{P}(S)$
- Cartesian Product: $S \times R$

See the B summary for formal definitions of these

New Sets from Old

- Union: $S \cup R$
- Intersection: $S \cap R$
- Powerset: $\mathbb{P}(S)$
- Cartesian Product: $S \times R$

See the B summary for formal definitions of these

New Sets from Old

- Union: $S \cup R$
- Intersection: $S \cap R$
- Powerset: $\mathbb{P}(S)$
- Cartesian Product: $S \times R$

See the B summary for formal definitions of these

Relations

- A **relation** $S \leftrightarrow R$ is a formal statement of the correspondence between elements of two sets
- Example: $CARS \leftrightarrow PRICE$
- A relation is a set of sets of ordered pairs:
 $S \leftrightarrow R = \mathbb{P}(S \times R)$
- Note the effect of the powerset: the set of relations $S \leftrightarrow R$ includes the empty set (no relation), the set of all ordered pairs ($S \times R$) (everything related to everything), as well as all subsets in between.
- the set of elements in the left of the ordered pairs is called the **domain**
- the set of elements in the right of the ordered pairs is called the **range**

Relations

- A **relation** $S \leftrightarrow R$ is a formal statement of the correspondence between elements of two sets
- **Example:** $CARS \leftrightarrow PRICE$
- A relation is a set of sets of ordered pairs:
 $S \leftrightarrow R = \mathbb{P}(S \times R)$
- Note the effect of the powerset: the set of relations $S \leftrightarrow R$ includes the empty set (no relation), the set of all ordered pairs ($S \times R$) (everything related to everything), as well as all subsets in between.
- the set of elements in the left of the ordered pairs is called the **domain**
- the set of elements in the right of the ordered pairs is called the **range**

Relations

- A **relation** $S \leftrightarrow R$ is a formal statement of the correspondence between elements of two sets
- Example: $CARS \leftrightarrow PRICE$
- A relation is a set of sets of ordered pairs:
 $S \leftrightarrow R = \mathbb{P}(S \times R)$
- Note the effect of the powerset: the set of relations $S \leftrightarrow R$ includes the empty set (no relation), the set of all ordered pairs ($S \times R$) (everything related to everything), as well as all subsets in between.
- the set of elements in the left of the ordered pairs is called the **domain**
- the set of elements in the right of the ordered pairs is called the **range**

Relations

- A **relation** $S \leftrightarrow R$ is a formal statement of the correspondence between elements of two sets
- Example: $CARS \leftrightarrow PRICE$
- A relation is a set of sets of ordered pairs:
 $S \leftrightarrow R = \mathbb{P}(S \times R)$
- Note the effect of the powerset: the set of relations $S \leftrightarrow R$ includes the empty set (no relation), the set of all ordered pairs ($S \times R$) (everything related to everything), as well as all subsets in between.
- the set of elements in the left of the ordered pairs is called the **domain**
- the set of elements in the right of the ordered pairs is called the **range**

Relations

- A **relation** $S \leftrightarrow R$ is a formal statement of the correspondence between elements of two sets
- Example: $CARS \leftrightarrow PRICE$
- A relation is a set of sets of ordered pairs:
 $S \leftrightarrow R = \mathbb{P}(S \times R)$
- Note the effect of the powerset: the set of relations $S \leftrightarrow R$ includes the empty set (no relation), the set of all ordered pairs ($S \times R$) (everything related to everything), as well as all subsets in between.
- the set of elements in the left of the ordered pairs is called the **domain**
- the set of elements in the right of the ordered pairs is called the **range**

Relations

- A **relation** $S \leftrightarrow R$ is a formal statement of the correspondence between elements of two sets
- Example: $CARS \leftrightarrow PRICE$
- A relation is a set of sets of ordered pairs:
 $S \leftrightarrow R = \mathbb{P}(S \times R)$
- Note the effect of the powerset: the set of relations $S \leftrightarrow R$ includes the empty set (no relation), the set of all ordered pairs ($S \times R$) (everything related to everything), as well as all subsets in between.
- the set of elements in the left of the ordered pairs is called the **domain**
- the set of elements in the right of the ordered pairs is called the **range**

Relation Examples

- No price information known: $\{\}$
- *barina* costs \$13990: $\{(barina, 13990)\}$
- *astra* costs \$25490: $\{(barina, 13990), (astra, 25490)\}$
- “optioned up” *barina* costs \$25490:
 $\{(barina, 13990), (barina, 25490), (astra, 25490)\}$
- Note that elements in both domain and range need not be unique.
- $dom(BULLET4) = \{barina, astra\}$
- $ran(BULLET4) = \{13990, 25490\}$

Relation Examples

- No price information known: $\{\}$
- *barina* costs \$13990: $\{(barina, 13990)\}$
- *astra* costs \$25490: $\{(barina, 13990), (astra, 25490)\}$
- “optioned up” *barina* costs \$25490:
 $\{(barina, 13990), (barina, 25490), (astra, 25490)\}$
- Note that elements in both domain and range need not be unique.
- $dom(BULLETT4) = \{barina, astra\}$
- $ran(BULLETT4) = \{13990, 25490\}$

Relation Examples

- No price information known: $\{\}$
- *barina* costs \$13990: $\{(barina, 13990)\}$
- *astra* costs \$25490: $\{(barina, 13990), (astra, 25490)\}$
- “optioned up” *barina* costs \$25490:
 $\{(barina, 13990), (barina, 25490), (astra, 25490)\}$
- Note that elements in both domain and range need not be unique.
- $dom(BULLETT4) = \{barina, astra\}$
- $ran(BULLETT4) = \{13990, 25490\}$

Relation Examples

- No price information known: $\{\}$
- *barina* costs \$13990: $\{(barina, 13990)\}$
- *astra* costs \$25490: $\{(barina, 13990), (astra, 25490)\}$
- “optioned up” *barina* costs \$25490:
 $\{(barina, 13990), (barina, 25490), (astra, 25490)\}$
- Note that elements in both domain and range need not be unique.
- $dom(BULLET4) = \{barina, astra\}$
- $ran(BULLET4) = \{13990, 25490\}$

Relation Examples

- No price information known: $\{\}$
- *barina* costs \$13990: $\{(barina, 13990)\}$
- *astra* costs \$25490: $\{(barina, 13990), (astra, 25490)\}$
- “optioned up” *barina* costs \$25490:
 $\{(barina, 13990), (barina, 25490), (astra, 25490)\}$
- Note that elements in both domain and range need not be unique.
- $dom(BULLETT4) = \{barina, astra\}$
- $ran(BULLETT4) = \{13990, 25490\}$

Relation Examples

- No price information known: $\{\}$
- *barina* costs \$13990: $\{(barina, 13990)\}$
- *astra* costs \$25490: $\{(barina, 13990), (astra, 25490)\}$
- “optioned up” *barina* costs \$25490:
 $\{(barina, 13990), (barina, 25490), (astra, 25490)\}$
- Note that elements in both domain and range need not be unique.
- $dom(BULLETT4) = \{barina, astra\}$
- $ran(BULLETT4) = \{13990, 25490\}$

Relation Examples

- No price information known: $\{\}$
- *barina* costs \$13990: $\{(barina, 13990)\}$
- *astra* costs \$25490: $\{(barina, 13990), (astra, 25490)\}$
- “optioned up” *barina* costs \$25490:
 $\{(barina, 13990), (barina, 25490), (astra, 25490)\}$
- Note that elements in both domain and range need not be unique.
- $dom(BULLETT4) = \{barina, astra\}$
- $ran(BULLETT4) = \{13990, 25490\}$

Functions

- special case of relations:
elements in domain are unique
- not all elements need be in domain: **partial**
- further special cases:
 - all elements in domain: **total**
 - elements in range unique: **injective**
 - all elements in range: **surjective**
- injective functions are also called **one-to-one**
- surjective functions are also called **onto**
- injective/surjective functions can be partial or total
- a function that is both injective and surjective is called **bijective**

Functions

- special case of relations:
elements in domain are unique
- not all elements need be in domain: **partial**
- further special cases:
 - all elements in domain: **total**
 - elements in range unique: **injective**
 - all elements in range: **surjective**
- injective functions are also called **one-to-one**
- surjective functions are also called **onto**
- injective/surjective functions can be partial or total
- a function that is both injective and surjective is called **bijective**

Functions

- special case of relations:
elements in domain are unique
- not all elements need be in domain: **partial**
- **further special cases:**
 - all elements in domain: **total**
 - elements in range unique: **injective**
 - all elements in range: **surjective**
- injective functions are also called **one-to-one**
- surjective functions are also called **onto**
- injective/surjective functions can be partial or total
- a function that is both injective and surjective is called **bijective**

Functions

- special case of relations:
elements in domain are unique
- not all elements need be in domain: **partial**
- further special cases:
 - all elements in domain: **total**
 - elements in range unique: **injective**
 - all elements in range: **surjective**
- injective functions are also called **one-to-one**
- surjective functions are also called **onto**
- injective/surjective functions can be partial or total
- a function that is both injective and surjective is called **bijective**

Functions

- special case of relations:
elements in domain are unique
- not all elements need be in domain: **partial**
- further special cases:
 - all elements in domain: **total**
 - elements in range unique: **injective**
 - all elements in range: **surjective**
- injective functions are also called **one-to-one**
- surjective functions are also called **onto**
- injective/surjective functions can be partial or total
- a function that is both injective and surjective is called **bijective**

Functions

- special case of relations:
elements in domain are unique
- not all elements need be in domain: **partial**
- further special cases:
 - all elements in domain: **total**
 - elements in range unique: **injective**
 - all elements in range: **surjective**
- injective functions are also called **one-to-one**
- surjective functions are also called **onto**
- injective/surjective functions can be partial or total
- a function that is both injective and surjective is called **bijective**

Functions

- special case of relations:
elements in domain are unique
- not all elements need be in domain: **partial**
- further special cases:
 - all elements in domain: **total**
 - elements in range unique: **injective**
 - all elements in range: **surjective**
- **injective functions are also called one-to-one**
- surjective functions are also called **onto**
- injective/surjective functions can be partial or total
- a function that is both injective and surjective is called **bijective**

Functions

- special case of relations:
elements in domain are unique
- not all elements need be in domain: **partial**
- further special cases:
 - all elements in domain: **total**
 - elements in range unique: **injective**
 - all elements in range: **surjective**
- injective functions are also called **one-to-one**
- surjective functions are also called **onto**
- injective/surjective functions can be partial or total
- a function that is both injective and surjective is called **bijjective**

Functions

- special case of relations:
elements in domain are unique
- not all elements need be in domain: **partial**
- further special cases:
 - all elements in domain: **total**
 - elements in range unique: **injective**
 - all elements in range: **surjective**
- injective functions are also called **one-to-one**
- surjective functions are also called **onto**
- **injective/surjective** functions can be partial or total
- a function that is both injective and surjective is called **bijective**

Functions

- special case of relations:
elements in domain are unique
- not all elements need be in domain: **partial**
- further special cases:
 - all elements in domain: **total**
 - elements in range unique: **injective**
 - all elements in range: **surjective**
- injective functions are also called **one-to-one**
- surjective functions are also called **onto**
- injective/surjective functions can be partial or total
- a function that is both injective and surjective is called **bijective**

Use of Functions and Relations for Data Modelling

- Since B does not have any data structures in the conventional sense, functions and relations are used
- For example, use function to model an array
- $array := \{(0, 45), (1, 23), (2, 16), (3, 18)\}$
- $baseprice := \{x \mapsto y \mid x \in HOLDENS \wedge y \in \mathbb{N}\}$
- $x \mapsto y$ is called a **maplet** (ordered pair)
- since the result of a function can itself be a function, can **use functions to model classes!**

Use of Functions and Relations for Data Modelling

- Since B does not have any data structures in the conventional sense, functions and relations are used
- For example, use function to model an array
- $array := \{(0, 45), (1, 23), (2, 16), (3, 18)\}$
- $baseprice := \{x \mapsto y \mid x \in HOLDENS \wedge y \in \mathbb{N}\}$
- $x \mapsto y$ is called a **maplet** (ordered pair)
- since the result of a function can itself be a function, can **use functions to model classes!**

Use of Functions and Relations for Data Modelling

- Since B does not have any data structures in the conventional sense, functions and relations are used
- For example, use function to model an array
- $array := \{(0, 45), (1, 23), (2, 16), (3, 18)\}$
- $baseprice := \{x \mapsto y \mid x \in HOLDENS \wedge y \in \mathbb{N}\}$
- $x \mapsto y$ is called a **maplet** (ordered pair)
- since the result of a function can itself be a function, can **use functions to model classes!**

Use of Functions and Relations for Data Modelling

- Since B does not have any data structures in the conventional sense, functions and relations are used
- For example, use function to model an array
- $array := \{(0, 45), (1, 23), (2, 16), (3, 18)\}$
- $baseprice := \{x \mapsto y \mid x \in HOLDENS \wedge y \in \mathbb{N}\}$
- $x \mapsto y$ is called a **maplet** (ordered pair)
- since the result of a function can itself be a function, can **use functions to model classes!**

Use of Functions and Relations for Data Modelling

- Since B does not have any data structures in the conventional sense, functions and relations are used
- For example, use function to model an array
- $array := \{(0, 45), (1, 23), (2, 16), (3, 18)\}$
- $baseprice := \{x \mapsto y \mid x \in HOLDENS \wedge y \in \mathbb{N}\}$
- $x \mapsto y$ is called a **maplet** (ordered pair)
- since the result of a function can itself be a function, can **use functions to model classes!**

Use of Functions and Relations for Data Modelling

- Since B does not have any data structures in the conventional sense, functions and relations are used
- For example, use function to model an array
- $array := \{(0, 45), (1, 23), (2, 16), (3, 18)\}$
- $baseprice := \{x \mapsto y \mid x \in HOLDENS \wedge y \in \mathbb{N}\}$
- $x \mapsto y$ is called a **maplet** (ordered pair)
- since the result of a function can itself be a function, can **use functions to model classes!**

Summary

- **Sets** are an essential component of the B Experience
- understand **Relations and Functions** as sets
- use Relations and Sets as **data structures and class modelling tools**

Summary

- **Sets** are an essential component of the B Experience
- understand **Relations and Functions** as sets
- use Relations and Sets as **data structures and class modelling tools**

Summary

- **Sets** are an essential component of the B Experience
- understand **Relations and Functions** as sets
- use Relations and Sets as **data structures and class modelling tools**