CS4211: Formal Methods for Software Engineering

Lecture Notes

Kevin Toh

Introduction to Formal Methods

- Requirements are difficult to define because its written in **Natural Language** which can be imprecise and ambiguous at times.
- As we cannot anticipate the ways a system may be used, written test cases only covers a small subset of use cases.
- We want to verify if a system always satisify a certain property, in possible all cases.
- In Formal Methods, we use **Mathematics** to define the **structure** and **behaviour** of our software because it is **precise** and **unambiguous**
- Eventually, we can use a model checker to automatically verify the software by checking if a certain property holds in all cases.

The Z Specification Language

- Based on set theory and mathematical logic
  - We will be doing a recap on predicates, set theory, functions and relations next.
- Uses schemas to declare object properties
  - Schemas are similar to defining the structure of a class and its properties in an Object-Oriented Programming Language.
- Uses operations to describe state transitions
  - Each object has a state representing the values it's properties hold at a certain moment in time.
  - Operations are similar to methods of a class.
  - Operations modify the state of an object.
  - We use predicates to describe state transitions in an operation.
- We can then proof that a certain property holds manually

Latex Setup

- The Latex package that we will be using is zed-csp
- **Setup Code:**

```
% For latex document
\documentstyle[12pt,zed]{article}

% For beamer slides
\usepackage{zed-csp}

\begin{document}
\end{document}
```

- Reference: https://sg.mirrors.cicku.me/ctan/macros/latex/contrib/zed-csp/zed2e.pdf

Recap on Predicates and Logic

Predicate

A statement that is either true or false.

1. There are 365 days in 2024. (**false**)
2. Let $P(x, y)$ be $x + y = 9$
   - $P(4, 5)$ is **true**.
   - $P(3, 7)$ is **false**.

Logic Operators

1. Not ($\neg$) Eg:
2. And ($\wedge$)
3. Or ($\vee$)
4. Implies ($\Rightarrow$)
5. Equivalence ($\Leftrightarrow$)

Recap on Quantifiers

1. Universal Quantifier ($\forall$)
   - Example: All natural numbers are greater than -1.
   - Mathematically, we would write $\forall n \in \mathbb{N}, n > -1$
   - In Z Specification, we would write $\forall n : \mathbb{N} \bullet n > -1$
   - $\forall n : \mathbb{N} \bullet n > 0$
   - In general, $\exists x : X \bullet P(x)$ abbreviates $P(a) \land P(b) \land P(c) \land \ldots$
2. Existential Quantifier ($\exists$)
   - Example: There exists a natural number more than 0.
   - In Z Specification, we would write $\exists n : \mathbb{N} \bullet n > 0$
   - In general, $\exists x : X \bullet P(x)$ abbreviates $P(a) \lor P(b) \lor P(c) \lor \ldots$

Differences between Mathematical Notations and Z Specification

- In Mathemical Notation, : or | means "such that" when used in Set Expressions.
- In Z Specification, : means "belongs to".
  - The difference between : and $\in$ will be explained later.
- In Z Specification, $\bullet$ means "such as" when writing predicates.

Recap on Set Theory

- A set is a collection of elements (or members)
  - Elements are not ordered: $\{a, b, c\} = \{b, a, c\}$
  - Elements are not repeated" $\{a, a, b\} = \{a, b\}$
  - Given Sets
    - $\mathbb{N} = \{0, 1, 2, 3, \ldots\}$ (The set of all natural numbers)
    - $\mathbb{N}_1 = \{1, 2, 3, \ldots\}$
    - $\mathbb{Z} = \{0, 1, -1, 2, -2, \ldots\}$ (The set of all integers)
    - $\mathbb{R}$ (The set of all real numbers)
    - $\emptyset$ (Empty Set: The set with no elements)
- Membership: $x \in \mathbb{X}$ is a predicate which is
  - true if x is in the set $\mathbb{X}$. Eg: $a \in \{a, b, c\}$
  - false if x is not in the set $\mathbb{X}$. Eg: $d \in \{a, b, c\}$

Difference between ':' and '$\in$'

Example: $\forall x : \mathbb{Z} \bullet x > 5 \Rightarrow x \in \mathbb{N}$

- $x : \mathbb{Z}$ declares a new variable x of type $\mathbb{Z}$
- $x \in \mathbb{N}$ is a predicate which is either true or false depending on the value of the declared x.

Recap on Set Theory

- Set Expressions
  - We can express a set by listing its elements if the set is finite and small.
    - $\{a, b, c, d\}$ is a finite set.
  - If a set is large or infinite, we can definite a set by giving a predicate which specifies precisely those elements in a set.
    - $\mathbb{N}$ is an infinite set.
    - The set of natural numbers less than 99 is $\{n : \mathbb{N} \mid n < 99\}$
    - In general the set $\{x : \mathbb{X} \mid P(x)\}$ is the set of elements of $\mathbb{X}$ for which predicate P is true.
- Set Examples
  - The set of even integers is $\{z : \mathbb{Z} \mid \exists\, k : \mathbb{Z} \bullet z = 2k\}$
  - The set of natural numbers which when divided by 7 leave a remaineder of 4 is $\{n : \mathbb{N} \mid \exists\, m : \mathbb{N} \bullet n = 7m + 4\}$
  - $\mathbb{N}$ is the set $\{z : \mathbb{Z} \mid z \geq 0\}$
  - $\mathbb{N}_1$ is the set $\{n : \mathbb{N} \mid n \geq 1\}$
  - If a, b are any natural numbers, then $a \,..\, b$ is defined as the set of all natural numbers between a and b inclusive.
    - $a \,..\, b$ is the set $\{n : \mathbb{N} \mid a \leq n \leq b\}$

Recap on Set Theory

- Subset ($\subseteq$): If S and T are sets, $S \subset T$ is a predicate equivalent to $\forall s : S \bullet s \in T$.
  - The following predicates are true:
    - $\{0, 1, 2\} \subseteq \mathbb{N}$
    - $2 \mathinner{..} 3 \subseteq 1 \mathinner{..} 5$
    - $\{a, b\} \subseteq \{a, b, c\}$
    - $\emptyset \subseteq X$ for any set X
    - $\{x\} \subseteq X \Leftrightarrow x \in X$
- Proper Subset ($\subset$): If S and T are sets, $S \subset T$ is a predicate equivalent to $S \subseteq T \land S \neq T$.
- Power Set ($\mathbb{P}$): If X is a set, $\mathbb{P}\ X$ (the power set of X) is the set of all subsets of X.
  - $A \in \mathbb{P}\ B = A \subseteq B$
  - The following predicates are true:
    - $\mathbb{P}\{a, b\} = \{\emptyset, \{a\}, \{b\}, \{a, b\}\}$
    - $\mathbb{P}\ \emptyset = \{\emptyset\} \neq \emptyset$
    - $1..5 \in \mathbb{P}\ \mathbb{N}$
    - $2..5 \in \mathbb{P}(1..5)$
  - If X has k elements, then $\mathbb{P}\ X$ has 2^k elements.

Recap on Set Theory

- Set Operations
  - Set Union: Suppose $S, T : \mathbb{P}\, X$ or $S \subseteq X, T \subseteq X$, then $S \cup T = \{x : X \mid x \in S \vee x \in T\}$
    - $\{a, b, c\} \cup \{b, g, h\} = \{a, b, c, g, h\}$
    - $A \cup \emptyset = A$ (for any set A)
  - Set Intersection: Suppose $S, T : \mathbb{P}\, X$, then $S \cap T = \{x : X \mid x \in S \wedge x \in T\}$
    - $\{a, b\} \cap \{b, c\} = \{b\}$
    - $\{a, b, c\} \cap \{d, g\} = \emptyset$ (disjoint sets)
    - $A \cap \emptyset = \emptyset$ (for any set A)
  - Set Difference: Suppose $S, T : \mathbb{P}\, X$, then $S - T = \{x : X \mid x \in S \wedge x \notin T\}$
    - $\{a, b, c\} - \{b, g, h\} = \{a, c\}$
    - $\mathbb{N}_1 = \mathbb{N} = \{0\}$
  - Cartesian Product: If A and B are sets, then $A \times B$ is the set of all ordered pairs (a, b) with $a \in A$ and $b \in B$.
    - $\{a, b\} \times \{a, c\} = \{(a, a), (a, c), (b, a), (b, c)\}$
  - Cardinality: $\#X$ is a natural number denoting the cardinality of (number of elements in) a finite set X.
    - $\#\{a, b, c\} = 3$

Tuples and Cartesian Product

- An n-tuple $(x_1, \ldots, x_n)$ is present in the Cartesian Product $\mathbf{a_1} \times \ldots \times \mathbf{a_n}$ if and only if each element x_i is an element of the corresponding set $\mathbf{a_i}$.
- To refer to a particular component of a tuple t, we use the projection notation $(.)$
- Suppose we have $t = (x_1, x_2, \ldots, x_n)$
  - The first component of the tuple t is written as $t.1$ which is the value $\mathbf{x_1}$.
  - The second component of the tuple t is written as $t.2$ which is the value $\mathbf{x_2}$.
  - The n-th component of the tuple t is written as $t.n$ which is the value $\mathbf{x_n}$

Recap on Relations

- A relation R from sets A to B, is declared as $R : A \leftrightarrow B$ is a subset of $A \times B$
- Example: $R = \{(c, x), (c, z), (d, x), (d, y), (d, z)\}$
  - The following predicates are equivalent
    1. $(c, z) \in R$
    2. $c \rightarrow z \in R$
    3. cRz
- **Domain:** $\mathrm{dom}\, R$ is the set $\{a : A \mid \exists\, b : B \bullet aRb\}$
- **Range:** $\mathrm{ran}\, R$ is the set $\{b : B \mid \exists\, a : A \bullet aRb\}$

Types in Z Specification

- Z specification language is **strongly typed**.
- Every expression is given a type.
- Any set can be used as a type.
- The following are equivalent declarations of variables x and y of types A and B respectively.
  - $(x, y) : A \times B$
  - $x : A, y : B$
  - $x, y : A$ (only when $B = A$)

Modelling using Z Specification

- When we write a program, we can write code procedurally, functionally or in an object oriented manner.
- Z Specification can help us model our code using two distinct sections.
  1. Declaration: To define variables.
  2. Predicate: Often used to define behaviours or invariants.
- **Example #1:** We can define the relation **divides** between two natural numbers.

  | `divides:` $\mathbb{N}_1 \leftrightarrow \mathbb{N}$

  | ----------

  | $\forall x : \mathbb{N}_1;\ y : \mathbb{N} \bullet x \ \underline{\text{divides}}\ y \Leftrightarrow \exists k : \mathbb{N} \bullet x \cdot k = y$

  Usage: 3 $\underline{\text{divides}}$ 6, $\neg$ (3 $\underline{\text{divides}}$ 7)
- **Example #2:** We can define the relation $\leq$ between two natural numbers.

  | $\_ <= \_$: $\mathbb{N} \leftrightarrow \mathbb{N}$

  | ----------

  | $\forall x, y : \mathbb{N} \bullet x <= y \Leftrightarrow \exists k : \mathbb{N} \bullet x + k = y$

  The relation $<=$ is the infinite subset of ordered pairs in $\mathbb{N} \times \mathbb{N}$.

  $\{(0,0), (0,1), (1,1), (0,2)(1,2), (2,2), \ldots\}$

Domain and Range Restriction

- Let A, B, R, S, T be sets.
- A is the domain set, B is the range set and R is the relation set.
- Note that S is a subset of the domain set and T is the subset of the range set.
- Suppose $R : A \leftrightarrow B, S \subseteq A$ and $T \subseteq B$.
  - **Domain Restriction:** $S \lhd R$ is the set $\{(a, b) : R \mid a \in S\}$
  - **Range Restriction:** $R \rhd T$ is the set $\{(a, b) : R \mid b \in T\}$
- Notice that both $S \lhd R \in A \leftrightarrow B$ and $R \rhd T \in A \leftrightarrow B$, meaning that both domain restriction and range restrictions are relations from sets A to B.
- If has_sibling: People $\leftrightarrow$ People then
  - female $\lhd$ has_sibling is the relationship is_sister_of.
  - has_sibling $\rhd$ female is the relationship has_sister.

Domain and Range Subtraction

- Let A, B, R, S, T be sets.
- A is the domain set, B is the range set and R is the relation set.
- Note that S is a subset of the domain set and T is the subset of the range set.
- Suppose $R : A \leftrightarrow B, S \subseteq A$ and $T \subseteq B$.
  - **Domain Subtraction:** $S \lhd R$ is the set $\{(a, b) : R \mid a \notin S\}$
  - **Range Subtraction:** $R \rhd T$ is the set $\{(a, b) : R \mid b \notin T\}$
- The following predicates are true.
  - $S \lhd R = (A - S) \lhd R$
  - $R \rhd T = R \rhd (B - T)$
  - $S \lhd R \in A \leftrightarrow B$
  - $R \rhd T \in A \leftrightarrow B$
- If has_sibling: People $\leftrightarrow$ People then
  - female $\lhd$ has_sibling is the relationship is_brother_of.
  - has_sibling $\rhd$ female is the relationship has_brother.

Relational Image

- Suppose the relation $R : A \leftrightarrow B$ and $S \leftrightarrow A$
- $R(\!| S |\!) = \{b : B \mid \exists a : S \bullet aRb\}$
- $R(\!| S |\!) \leftrightarrow B$
- Example
  - $divides(\!| \{8, 9\} |\!) = \{x : \mathbb{N} \mid \exists k : \mathbb{N} \bullet x = 8 \cdot k \lor 9 \cdot k\} = \{0, 8, 9, 16, 18, \ldots\}$
  - $<= (\!| 3, 7, 21 |\!) = \{x : \mathbb{N} \mid x >= 3\}$
- In summary, the relational image returns the set of all elements $b \in B$ such that there exists an $a \in S$ with $(a, b) \in R$.
- The difference between relational image and range restriction is that range restriction returns the subset of R which are ordered pairs of (a, b) where $a \in A$ and $b \in B$ and the first element a of the ordered pair is in S. The relational image simply just returns the set of all second elements b.

Inverse and Relational Composition

- **Inverse:** R^{-1} is the set $\{(b, a) : B \times A \mid aRb\}$ or $R^{-1} \in B \leftrightarrow A$
  - $has\_sibling^{-1} = has\_sibling$
  - $divisor^{-1} = has\_divisor$
- Example: $succ^{-1} = \text{pred}$

  | succ: $\mathbb{N} \leftrightarrow \mathbb{N}$

  |----------

  | $\forall x, y : \mathbb{N} \bullet x \ \underline{\texttt{succ}} \ y \leftrightarrow x + 1 = y$
- **Relational Composition ($\fatsemi$)**
  - Suppose $R : A \leftrightarrow B$ and $S : B \leftrightarrow C$ are two relations.
  - $R \fatsemi S = \{(a, c) : A \times C \mid \exists\, b : B \mid aRb \land bSc\}$
  - $R \fatsemi S \in A \leftrightarrow C$
- Examples
  - $is\_parent\_of \fatsemi is\_parent\_of = is\_grandparent\_of$
  - $R^0 = id[A]$
  - $R^1 = R$
  - $R^2 = R \fatsemi R$
  - $R^3 = R \fatsemi R \fatsemi R$

Recap on Functions

- A (partial) function from a set A to a set B, denoted by $f : A \pfun B$ is a subset f of $A \times B$ with the property that for each $a \in A$, there is **at most one** $b \in B$ with $(a, b) \in f$.
- $\dom f$ is the set $\{a : A \mid \exists\, b : B \bullet (a, b) \in f\}$
- $\ran f$ is the set $\{b : B \mid \exists\, a : A \bullet (a, b) \in f\}$
- Suppose $f : A \pfun B$ and $a \in \dom f$, then $f(a)$ denotes the unique image $b \in B$ that a is mapped to by f.
- $(a, b) \in f$ is equivalent to $f(a) = b$
- **Total Function:** If the function $f : A \pfun B$ is a total function, then $f : A \fun B$ if and only if $\dom f = A$

Function Overriding

- Suppose $f, g : A \pfun B$, then $f \oplus g$ is the function $(\operatorname{dom} g \domsub f) \cup g$.
- The following predicates are true:
  1. $\operatorname{dom} f \oplus g = \operatorname{dom} f \cup \operatorname{dom} g$
  2. $a : \operatorname{dom} g \bullet (f \oplus g)(a) = g(a)$
  3. $\forall a : \operatorname{dom} f - \operatorname{dom} g \bullet (f \oplus g)(a) = f(a)$
  4. $\mathrm{f} \oplus \mathrm{g} \in a \pfun b$
- Examples
  1. $\{a \to x, b \to y, c \to x\} \oplus \{a \to y\} = \{a \to y, b \to y, c \to x\}$
  2. double $\oplus$ root $= \{(0,0), (1,1), (2,4), (3,6), (4,2), \ldots\}$ (Note: $(4,8)$ was replaced with $(4,2)$ as the domain 4 is both in f and g, so the range was replaced with 2 that was in g)

Specifying Functions

1. Using a look-up table
   - If a function $f : A \pfun B$ is finite (and not too large), we can specify the function explicity by listing all pairs (a, b) in the subset $A \times B$.
   - Example: PassportNo $\to$ Address

PassportNo	Address
A001017	77 Sunset Strip
...	...
G707165	19 Mail Street

2. Declaring Axioms: A function can be specified by giving a **predicate** determining which pairs (a, b) are in the function.
   - **Example: The root function that calculates the square root of a natural number**

     | `root` $\mathbb{N} \pfun \mathbb{N}$
     |---
     | $\mathrm{dom}\, root = \{n : \mathbb{N} \mid \exists m : \mathbb{N} \bullet m^2 = n\}$
     | $\forall n : \mathrm{dom}\, root \bullet (root(n))^2 = n$

Specifying Functions

3. Using Recursion: For functions defined recursively in terms of itself.

   | fact $\mathbb{N}_1 \nrightarrow \mathbb{N}$
   |---
   | fact(1) = 1
   | $\forall n : \mathbb{N}_1 - \{1\} \bullet fact(n) = n * fact(n-1)$

4. Giving an Algorithm: A function $f : A \nrightarrow B$ is specified by an algorithm such that given any element a in the domain of f, the element $f(a)$ can be computed using the algorithm.

```
1  input n : N
2  var x, y: integer;
3  begin
4      x := n; y:= 0;
5      while x != 0 do
6      begin
7          x := x - 1; y:= y + 2
8      end;
9  write(y)
10 end
```

| double: $\mathbb{N} \rightarrow \mathbb{N}$
| ---
| $\forall n : \mathbb{N} \bullet double(n) = 2n$

Sequences

- A sequence s of elements of a set A, denoted $s : \mathrm{seq}\, A$, is a function $s : \mathbb{N} \pfun A$ where $\mathrm{dom}\, s = 1 \mathinner{..} n$ for some natural number n.
- **Example**
  - $< b, c, a, b >$ denotes the sequence (function) $\{1 \mapsto b, 2 \mapsto c, 3 \mapsto a, 4 \mapsto b\}$
  - The empty sequence is denoted by $<>$
- The set of all sequences of elements from A is denoted as $\mathrm{seq}\, A$ and is defined to be $\mathrm{seq}\, A = \{s : \mathbb{N} \pfun A \mid \exists n : \mathbb{N} \bullet \mathrm{dom}\, s = 1 \mathinner{..} n\}$
- $\mathrm{seq}_1\, A = \mathrm{seq}\, A - \{<>\}$ is defined as the set of non-empty sequences.
- Since sequences are ordered mapping, $< a, b, a > \neq < a, a, b > \neq < a, b >$

Special Functions for Sequence

1. Concatenation
   - $< a, b > \frown < b, a, c > = < a, b, b, a, c >$
2. Head
   - $\mid$ `head:` $\quad \mathrm{seq}_1 A \rightarrow A$
     $\mid$`----------`
     $\mid \forall s : \mathrm{seq}_1 A \bullet head(s) = s(1)$
   - $head < c, b, b > = c$
3. Tail
   - $\mid$ `tail:` $\quad \mathrm{seq}_1 A \rightarrow \mathrm{seq}\, A$
     $\mid$`----------`
     $\mid \forall s : \mathrm{seq}_1 A \bullet < head(s) > \frown tail(s) = s$
   - $tail < c, b, b > = < b, b >$
4. Filter
   - $< a, b, c, d, e, d, c, b, a > \upharpoonright \{a, d\} = < a, d, d, a >$
   - Filter only keeps the element in the specified set, preserves order in the original sequence and outputs a new sequence.

Z Specification

- As explained in an earlier slide, we can write Z Specification to formally specify requirements in **two distinct sections**
  1. Declaration: To define variables.
  2. Predicate: Often used to restrain the possible values of the declared variables to define behaviours or invariants.
- Formal Specification can be done in the form of both **axioms** and **schemas**.
- When we were specifying functions earlier, we were formally specifying them in the form of **axioms** in the **axiom environment**.
- Later we will introduce how to formally specify **schemas** which are used to specify relationships between variable values.
- There are two main type of schemas in the **schema environment**
  - State Schema
  - Operation Schema

The zed-csp Package

- We can use the zed-csp package to formally specify requirements in Z Specification and render axioms and schemas them in TeX.
- There are **two types of environment** in the zed-csp package.

1. Axiom Environment

$$limit : \mathbb{N}$$
$$\hline$$
$$limit \leq 65535$$

```
\begin{axdef}
    limit: \nat
    \where
    limit \leq 65535
\end{axdef}
```

2. Schema Environment

PhoneDB

$$known : \mathbb{P}\, NAME$$
$$phone : NAME \pfun PHONE$$
$$\hline$$
$$known = \mathrm{dom}\, phone$$

```
\begin{schema}{PhoneDB}
known: \power NAME \\ phone: NAME \pfun PHONE
\where
known = \dom phone
\end{schema}
```

State Schema

- A **state schema** specifies a relationship between variable values
- It specifies a snapshot of a system
- **Variables** are declared and typed in the **top part of the schema**.
- A **predicate (axiom)** restraining the possible values of the declared variables are given in the **bottom part of the schema.**
- An instance of a schema is an assignment of values to variables consistent with their type declaration and satisfying the predicate.

Operation Schema

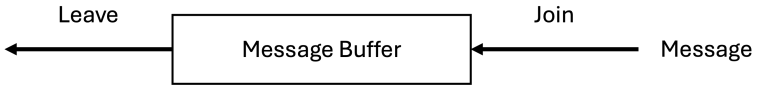
- The state schema provides a static view of the system.
- To specify how the system can change, we need to specify the operation schema.
- An operation can be thought as taking an instance of the state schema and producing a new instance.
- To specify such an operation, we express as a predicate the relationship between the **instance of the state before the operation** and the **instance after the operation.**

Convention

- The value of the state variables before the operation are denoted by **unprimed identifiers**.
  - Example: `items : seq MSG`
- Values after the operation are denoted by **primed identifiers**.
  - Example: `items' : seq MSG`
- Note: Let `MSG` be the set of all possible messages that can be transmitted.

Case Study: A Message Buffer

- We are going to model a message buffer to learn about **schema specification**.
- A message buffer stores messages within a queue data structure and operates on a first in / first out (FIFO) principle.
- Suppose we have a line which might be occupied by traffic, many messages join the buffer and but only the first message in the queue leaves the buffer when the line is free.
- The buffer may contain several messages at any time, but there is a fixed upper limit on the number of messages the buffer may contain.
- To model a message buffer, we minimally need the following:
  1. States: Buffer (to store messages)
  2. Operations: Join (new message joins the buffer), Leave (message leaves the buffer)



Formal Specification: Message Buffer

State Schema: Buffer

- Let MSG be the set of all possible messages that can be transmitted.
- Let max : $\mathbb{N}$ be the constant maximum number of messages that can be held in the buffer at any one time.
- Example: Let $MSG = \{m1, m2, m3\}$ and $max = 4$
  1. $items =< m1, m2 >$ is a valid instance.
  2. $items =< m3, m1, m1, m2, m2 >$ is an invalid instance.

Operation Schema: Join

- The decoration ? denotes an input
- There is an implicit $\wedge$ between each line in the predicate section.

Buffer

$items : \operatorname{seq} MSG$

$\#items \leq max$

Join

$items, items' : \operatorname{seq} MSG$
$msg? : MSG$

$\#items \leq max$
$\#items' \leq max$
$\#items < max$
$items' = items^\frown < msg? >$

Explanation: Predicate of the Join Operation

$$
\begin{array}{|l}
\textit{Join} \\
\hline
\textit{items}, \textit{items}' : \operatorname{seq} \textit{MSG} \\
\textit{msg}? : \textit{MSG} \\
\hline
\#\textit{items} \leq \textit{max} \\
\#\textit{items}' \leq \textit{max} \\
\#\textit{items} < \textit{max} \\
\textit{items}' = \textit{items}^\frown < \textit{msg}? > \\
\hline
\end{array}
$$

- The first two lines of the predicate indicate that we have a valid instance of the state schema **Buffer** both before and after the operation.
- The third line of the predicate is a pre-condition for the operation. It indicates that for the **Join** operation to be possible, the buffer must not be completely full.
- The last line of the predicate specifies the relationship between the buffer contents before and after the operation which is that the input message is already appended to the sequence of messages already in the buffer.

Formal Specification: Message Buffer (Continued)

Operation Schema: Leave

- The decoration ! denotes an output
- There is an implicit $\wedge$ between each line in the predicate section.
- Explanation for Leave Operation Predicate
  1. The first two lines of the predicate indicate that we have a valid instance of the state schema **Buffer** both before and after the operation.
  2. The third line of the predicate is a **pre-condition** for the operation. It indicates that for the **Leave** operation to be possible, the buffer must not be empty.
  3. The last line of the predicate specifies the relationship between the buffer contents before and after the operation. The output message is taken from the head of the sequence of messages in the buffer, leaving just the tail of the sequence of buffers.

Leave

$items, items' : \mathrm{seq}\, MSG$

$msg! : MSG$

$\#items \leq max$

$\#items' \leq max$

$\#items \neq \emptyset$

$items = < msg! > \frown items'$

Special States: Delta (Δ) and Initial State ($_{INIT}$)

1. Delta (Δ): To specify a **before** and **after** instance of the state schema for any operation.

$$\Delta Buffer$$
$$items, items' : \operatorname{seq} MSG$$
$$\#items \leq max$$
$$\#items' \leq max$$

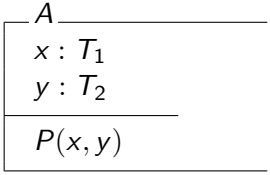
2. Initial State ($_{INIT}$): To specify a state when an instance of a state is first initialized.

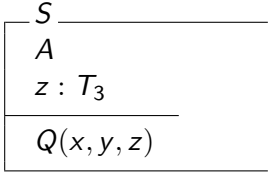
$$Buffer_{INIT}$$
$$Buffer$$
$$items = <>$$

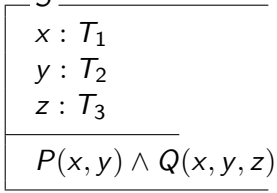
- Initially the buffer would be empty.
- Then, the operations of **Join** and **Leave** can occur whenever they are enabled.
- Operations are assumed to be atomic.
- At all times, an observer will notice that the state schema is satisfied.

Schema Inclusion

- Schema Inclusion is the act of including a schema in the declaration of another schema.
- It means the included schema has its declaration added to the new schema, and its predicate cojoined to the predicate of the new schema.
- The first "S" Schema is the **short form**, while the second "S" Schema is the **long form**.







Example: Schema Inclusion

Join

$\Delta Buffer$

$msg? : MSG$

$\#items < max$

$items' = items^\frown < msg? >$

Leave

$\Delta Buffer$

$msg! : MSG$

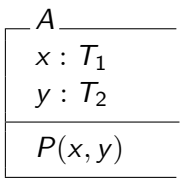
$items \neq \emptyset$

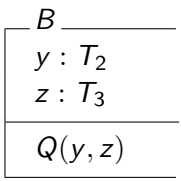
$items =< msg! > ^\frown items'$

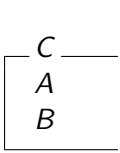
- We can include the $\Delta Buffer$ schema to both the Join and Leave operations.
- Explanation for Leave Operation Predicate
  1. The first line of the predicate is a **pre-condition** for the operation. It indicates that for the **Leave** operation to be possible, the buffer must not be empty.
  2. The last line of the predicate specifies the relationship between the buffer contents before and after the operation. The output message is taken from the head of the sequence of messages in the buffer, leaving just the tail of the sequence of buffers.

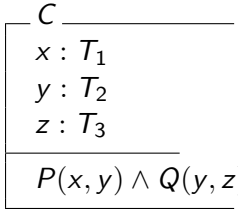
Merging Schemas

- **Type Compatability** is needed to merge schemas.
- In this case, the variable y is common between states A and B.
- We can simply merge the two types into a new state C without further specifying any new predicates.
- The full form of state C is also provided.









Extending Specifications: Slow Buffer and Slow Operations

- Applying concepts such as **Schema Inclusion** and **Merging Schemas**, we can extend specifications similar to inheritance or creating specialized classes in Object-Oriented Programming.
- To demonstrate this, we will attempt to model a **Slow Buffer** which has a constant $delay : \mathbb{N}$ to simulate that each new message can only join the buffer after $delay$ seconds.

SlowBuffer

$Buffer$

$idle : \mathbb{N}$

Tick

$\Delta SlowBuffer$

$idle' = idle + 1$

$items' = items$

$SlowBuffer_{INIT}$

$SlowBuffer$

$Buffer_{INIT}$

$idle = 0$

SlowJoin

$\Delta SlowBuffer$

$Join$

$idle \geq delay$

$idle' = 0$

SlowLeave

$\Delta SlowBuffer$

$Leave$

$idle \geq delay$

$idle' = 0$

Extending Specifications: Slow Buffer and Slow Operations (Full Form)

SlowBuffer

$items : \mathrm{seq}\ MSG$
$idle : \mathbb{N}$

$\#items \leq max$

$SlowBuffer_{INIT}$

$items : \mathrm{seq}\ MSG$
$idle : \mathbb{N}$

$\#items \leq max$
$items = <>$
$idle = 0$

SlowJoin

$items, items' : \mathrm{seq}\ MSG$
$idle, idle' : \mathbb{N}$
$msg? : MSG$

$\#items \leq max \land \#items' \leq max$
$\#items < max \land items' = items^\frown < msg? >$
$idle \geq delay \land idle' = 0$

SlowLeave

$items, items' : \mathrm{seq}\ MSG$
$idle, idle' : \mathbb{N}$
$msg! : MSG$

$\#items \leq max \land \#items' \leq max$
$items \neq \emptyset \land items = items'^\frown < msg? >$
$idle \geq delay \land idle' = 0$

Tick

$items : \mathrm{seq}\ MSG$
$idle : \mathbb{N}$

$\#items \leq max \land \#items' \leq max$
$idle' = idle + 1 \land items' = item$

Reasoning About The Specification

- Suppose, we want to verify that message buffer specified has the **FIFO property**.
- We want to show that the messages leave the buffer in the same order they arrive.
- In this case, we introduce **auxillary sequences** `inhist` and `outhist` to record the history of the flow of messages into and out of the buffer.
- Create a new schema which includes the original Buffer and Operation schemas and extra information about the auxillary variables.
- When a message **joins** the buffer, it is also added to the `inhist` sequence.
- When a message **leaves** the buffer, it is added to the `outhist` sequence.

Recorded Buffer and Operations

RecordedBuffer

$Buffer$

$inhist : \mathrm{seq}\ MSG$

$outhist : \mathrm{seq}\ MSG$

RecordedJoin

$\Delta RecordedBuffer$

$Join$

$inhist' = inhist^\frown < msg? >$

$outhist' = outhist$

RecordedBuffer$_{INIT}$

$RecordedBuffer$

$Buffer_{INIT}$

$inhist = <>$

$outhist = <>$

RecordedLeave

$\Delta RecordedBuffer$

$Join$

$inhist' = inhist$

$outhist' = outhist^\frown < msg! >$

RecordedJoin: Expanded Schema

RecordedJoin

$items, items' : \operatorname{seq} MSG$

$inhist, inhist' : \operatorname{seq} MSG$

$outhist, outhist' : \operatorname{seq} MSG$

$msg? : MSG$

$\#items \leq max \land \#items' \leq max$

$\#items < max \land items' = items^\frown < msg? >$

$inhist' = inhist^\frown < msg? > outhist' = outhist$

Proving the FIFO Property

- How can we use the auxillary variables `inhist` and `outhist` to prove that the buffer satisifies the **FIFO property**?
- We can prove that the predicate $\forall\, RecordedBuffer \bullet inhist = outhist \frown items$ is true.
- Prove using structural induction.
  1. Initially `inhist = outhist = items = <>`, so the predicate is true for the initial state.
  2. Suppose the predicate is true, and RecordedJoin occurs. After the operation, $inhist' = inhist \frown < msg? > \land outhist' = outhist \land items' = items \frown < msg? >$
  3. Hence, $inhist \frown < msg? > = (outhist \frown items) \frown < msg? >$ $= outhist \frown (items \frown < msg? >) = outhist' \frown items'$
  4. Therefore, the predicate remains true.
- We can construct a similar argument that the operation **RecordedLeave** also preserves the predicate.

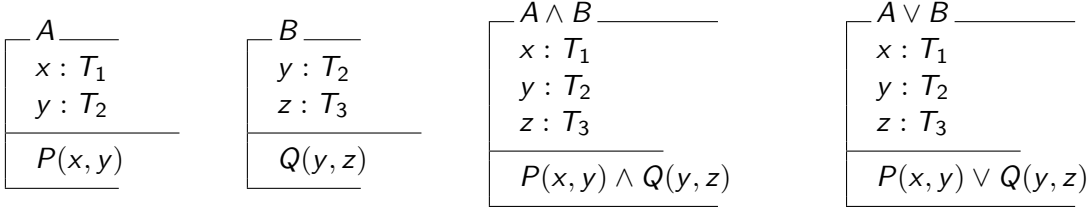
Conjunction of Schemas

- When using the conjunction ($\wedge$) operator on two schemas, it is equivalent to merging the two schemas.
- Suppose A and B are schemas
  - The declaration of $A \wedge B$ is the **union** of the declarations of A and B
  - The predicate of $A \wedge B$ is the **conjunction** of the predicates of A and B
- Examples
  1. $SlowRecordedBuffer \mathrel{\widehat=} SlowBuffer \wedge RecordedBuffer$
  2. $SlowRecordedBuffer_{INIT} \mathrel{\widehat=} SlowBuffer_{INIT} \wedge RecordedBuffer_{INIT}$
  3. $SlowRecordedJoin \mathrel{\widehat=} SlowJoin \wedge RecordedJoin$
- SlowRecordedBuffer Schema

$$\begin{array}{|l} \hline SlowRecordedBuffer \\ \hline SlowBuffer \\ RecordedBuffer \\ \hline \end{array}$$

Disjunction of Schemas

- Using the conjunction ($\wedge$) operator on two schemas yields a different result.
- Suppose A and B are schemas
  - The declaration of $A \vee B$ is the **union** of the declarations of A and B
  - The predicate of $A \vee B$ is the **disjunction** of the predicates of A and B



Disjunction of Schemas: Example

- Let $Flag ::= ok \mid error$ (The Flag type can be either 'ok' or 'error')
- Ξ State is another special state that is used for operations that access information in the state **without changing the state at all**.
- Example: $CompleteJoin \mathrel{\widehat=} JoinOk \lor JoinError$

JoinOk

$Join$

$flag! : Flag$

$flag! = ok$

JoinError

$\Xi Buffer$

$flag! : Flag$

$\#items = max \land flag! = error$

CompleteJoin

$\Delta Buffer$

$msg? : MSG;\ flag! : Flag$

$\#items < max \land items' = item^\frown < msg? > \land flag! = ok$

$\lor$

$\#items = max \land items' = items \land flag! = error$

Composition of Schemas

- Using the composition operator ($\fatsemi$) on two schemas is typically used to combine the effects of two operations.
- Example: $JoinLeave = Join \fatsemi Leave$
  - The pre-state of *Join* is the pre-state of $Join \fatsemi Leave$.
  - The post-state of *Join* is identified with the pre-state of *Leave* hidden within $Join \fatsemi Leave$.
  - The consequent post-state of *Leave* is the post-state of $Join \fatsemi Leave$.
- Convention: Hidden state is denoted with double prime (″).

JoinLeave

$\Delta Buffer$

$msg?, msg! : MSG$

$\#items < max$

$\exists items'' : \mathrm{seq}\, MSG \bullet items'' = items^\frown < msg? > \wedge items'' = < msg! > ^\frown items'$

Composition of Schemas in general

A

$x : T_1$

$y : T_2$

$P(x, y)$

AOP_1

δA

$t_3? : T_3;\ t_4! : T_4$

$Q_1(x, x', y, y', t_3?, t_4!)$

AOP_2

δA

$t_5? : T_5;\ t_6! : T_6$

$Q_2(x, x', y, y', t_5?, t_6!)$

$AOP_1 \fatsemi AOP_2$

δA

$t_3? : T_3;\ t_4! : T_4;\ t_5? : T_5;\ t_6! : T_6$

$\exists x'' : T_1;\ y'' : T_2 \bullet Q_1(x, x', y, y', t_3?, t_4!) \land Q_2(x, x', y, y', t_5?, t_6!)$