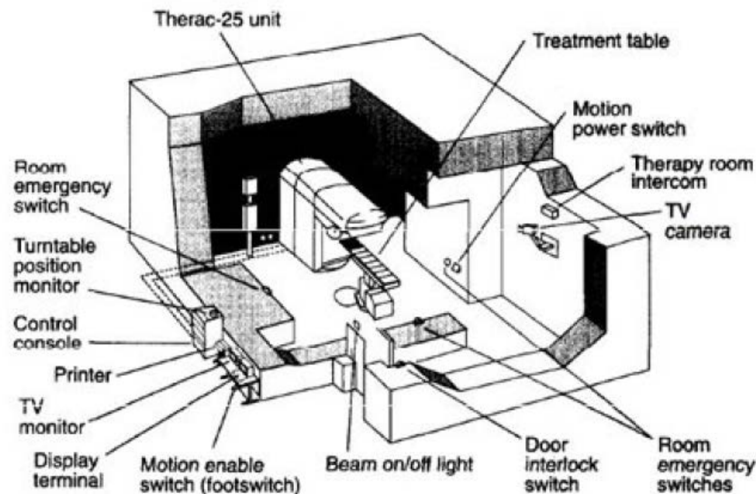
**Ariane 5 explosion**
Cost: $7 billion

**Intel Pentium 1994 Bug**
Cost: $800 million

4195835 – 4195835 / 3145727*3145727 = 256 ?

# Importance of Formal Specification & Design

**Therac-25 Radiation Overdose (1985-87)**
**3 patients** died




**Proton Therapy Machine Overdose (2014)**
Source: **still unknown**

BBC

News | Sport | Weather | Earth | Future | Sh

# NEWS UK

Home | World | Asia | Australia | India | China | UK | Business | Health | Science/Environment | Technology

England | Northern Ireland | Scotland | Wales | UK Politics | Education

**12 December 2014** Last updated at 15:28

## London airspace closed after computer failure

**London airspace has been closed until 19:00 GMT after a computer failure, air traffic controllers have said.**

The news was announced in a brief message on flight safety body Eurocontrol's website.

**“How can we provide people with software systems we can all depend on?"**

# Early Work on Formal Analysis

- 1949 Alan Turing: "Checking Large Routine"

- 1978 C. A. R. Hoare (1980 Turing Winner). Communicating Sequential Processes: event based calculus for modelling concurrency and communication. Promoter on Z formal specification language.

- 2007 three researchers won ACM Turing Award for inventing model checking, one of them E. Clarke has also won Franklin Institute 2014 Bower Award.
  - successfully applied in industries, e.g.,

# Formal Specification and Design Techniques (CS5232)

**Dr. DONG Jin Song**

thank Dr. R. Duke and Prof. G. Rose for the joint work on Object-Z and for providing some parts of the notes

thank Dr. B. Mahony for the joint work on TCOZ

**http://www.comp.nus.edu.sg/~ dongjs/ic52z5.html**

## Course Overview

- Introduction to Set, Logic and Z Notation
- Object-Oriented Z — Object-Z
- Advanced Formal Object Modelling Techniques
- Event Based Formalism — CSP
- Process Analysis Toolkit (PAT)
- Timed Communicating Object Z (TCOZ)

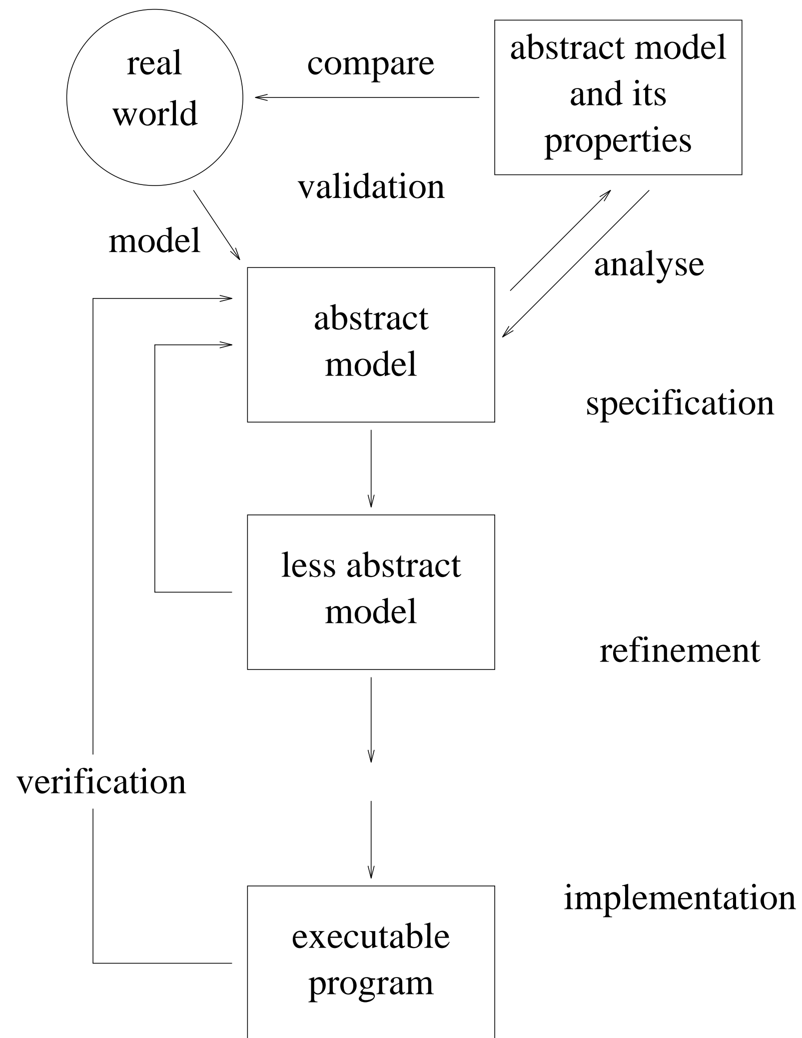
# Part 1 — Introduction and Background

## The Classical Engineer

- models with calculus, geometry
- analyses using classical theorems (Newton, Fourier, Gauss, ...)
- constructs a *hard* product

## The Software Engineer

- models with set theory, logic
- analyses using rules of inference
- constructs a *soft* product


real
world
compare
abstract model
and its
properties
validation
model
analyse
abstract
model
specification
less abstract
model
refinement
verification
implementation
executable
program

## Why Formal Specification?

A formal specification should

- add clarity and understanding by giving a description of the system which is
  - complete
  - unambiguous
  - easily analysed;
- lead to better code that is
  - reliable
  - accurate
  - maintainable
  - reusable
  - verified.

## Formal Specification and Software Engineering

(Towards an integrated methodology for software engineering.)
Formal specification

- is not a replacement, but rather an enhancement of existing methodologies;
- can only be effective if integrated within an overall methodology for software engineering.
- Implications of Using Formal Specification
  - training in the use of notation
  - integration with informal methodologies
  - translation for client consumption
  - emphasis upon abstraction

Formal methods by themselves will not solve the software crisis.

## A Formal Specification forms

the basis of the

- contract between designer and client
- plan for builder (programmer)
- reference document for
  - disputes, verification (testing)
  - user documentation, maintenance

## A Formal Specification should be

- precise
- readily analysed
- easy to modify, refine
- understandable

## A Specification Language should

- be a precise thinking tool for designers
- enhance communication between
  - designers
  - designer and programmer
  - designer and client
- enable formal (rigorous) analysis
- lead to separation of concerns
- allow abstraction, non-determinism
- encourage a sound design style
- have a formal syntax and semantics
- be usable

## Some Specification Languages

**State Oriented**

Systems modelled by an underlying state which can undergo change:

VDM, Z, Object-Z

**Process Algebra**

Systems modelled as processes partaking in communication:

CSP, CCS, LOTOS

**Algebraic**

Systems modelled by equations related by axioms (re-writing rules):

ACT 1, CLEAR, OBJ, Larch

## The Z Specification Language

- developed originally at Programming Research Group, Oxford University
- based on set theory and predicate logic
- system described by introducing fixed sets and variables and specifying the relationships between them using predicates
- declarative, not procedural
- system state determined by values taken by variables subject to restrictions imposed by state invariant
- operations expressed by relationship between values of variables before, and values after, the operation
- variable declarations and related predicates encapsulated into schemas
- schema calculus facilitates the composition of complex specifications

## Predicate Calculus

A predicate (proposition) is a statement that is either true or false.

- today is Monday
- $x + y = 9$ $P(x, y)$
- Logic operators:
  - Not ($\neg$), e.g. $\neg\,(11 < 3)$ is true
  - And ($\wedge$), e.g. $(11 > 3) \wedge (2 + 2 = 4)$
  - Or ($\vee$), e.g. $P \vee (\neg\, P)$ (a **tautology**)
  - Implies ($\Rightarrow$), e.g. $(11 < 3) \Rightarrow (2 + 2 = 5)$ is true
  - Equivalence ($\Leftrightarrow$), e.g. $P \Leftrightarrow P$ (is a tautology)

## Universal Quantifier ($\forall$)

Consider the predicate

"all natural numbers are bigger than zero".

We can write this formally as

$$\forall \quad n \quad : \quad \mathbb{N} \quad \bullet \quad n > 0$$

More generally,

$\forall x : X \bullet P(x)$ abbreviates $P(a) \land P(b) \land P(c) \land \cdots$

Are the following predicates true of false?

$$\forall n : \mathbb{N} \bullet n^2 > n$$

$$\forall n : \mathbb{N} \bullet (n^2 = n) \;\Rightarrow\; (n = 0 \;\lor\; n = 1)$$

## Existential Quantifier ($\exists$)

Consider the predicate

"there is a natural number bigger than zero".

We can write this formally as

$$\exists \quad n \quad : \quad \mathbb{N} \quad \bullet \quad n > 0$$

More generally,

$\exists\, x : X \bullet P(x)$ abbreviates $P(a) \vee P(b) \vee P(c) \vee \cdots$

Are the following predicates true or false?

$$\exists\, x : \mathbb{N} \bullet x = x + 1$$

$$\forall\, x : \mathbb{N} \bullet (\exists\, y : \mathbb{N} \bullet y > x)$$

# Sets

A set is a collection of elements (or members). e.g.

$$\{a, b, c\}, \quad \{3, 1, 16\}$$

- the elements are not ordered

$$\{a, b, c\} \text{ is the same set as } \{b, a, c\}$$

- the elements are not repeated

$$\{a, a, b\} \text{ is the same set as } \{a, b\}$$

**Some Given Sets**

$\mathbb{N} == \{0, 1, 2, \ldots\}$ natural numbers
$\mathbb{N}_1 == \{1, 2, 3, \ldots\}$
$\mathbb{Z} == \{0, 1, -1, 2, -2, \ldots\}$ integers
$\mathbb{R}$ real numbers
$\varnothing$ empty set: the set with no elements

## Membership

$$x \in X$$

is a predicate which is

- true if $x$ is in the set $X$, e.g $a \in \{a, b, c\}$ (T)
- false if $x$ is not in the set $X$, e.g $d \in \{a, b, c\}$ (F)

Notice the difference between ':' and '$\in$':

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

$x : \mathbb{Z}$ declares a new variable $x$ of type $\mathbb{Z}$

$x \in \mathbb{N}$ is a predicate which is true or false depending upon the value of the previously declared $x$

## Set Expressions

$\{a, b, c, d\}$ (is a *finite* set)
$\mathbb{N}$ (is an *infinite* set)

We can express a set by listing its elements, but this is impractical if the set is large, and impossible if the set is infinite.

Instead, a set can be defined by giving a predicate which specifies precisely those elements in the set.

e.g. the set of all natural numbers less than 99 is:

$$\{ \quad n : \mathbb{N} \quad \mid \quad n < 99 \quad \}$$

In general, the set

$$\{x : X \mid P(x)\}$$

is the set of elements of $X$ for which the predicate $P$ is true.

## 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 remainder of 4 is

$$\{n : \mathbb{N} \mid \exists\, m : \mathbb{N} \bullet n = 7m + 4\}$$

$\mathbb{N}$ is the set $\{z : \mathbb{Z} \mid z \geqslant 0\}$

$\mathbb{N}_1$ is the set $\{n : \mathbb{N} \mid n \geqslant 1\}$

if $a, b$ are any natural numbers then $a \mathinner{..} b$ is defined as the set of all natural numbers between $a$ and $b$ inclusive, i.e.

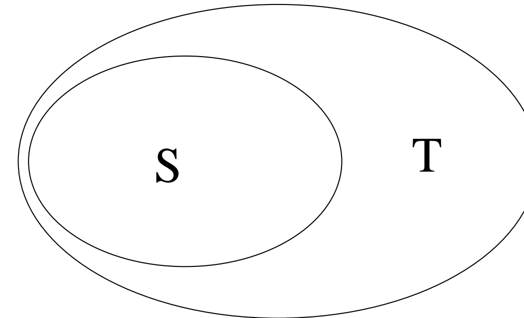
$a \mathinner{..} b$ is the set $\{n : \mathbb{N} \mid a \leqslant n \leqslant b\}$

## **Subset** (⊆) and **Proper Subset** (⊂)

If $S$ and $T$ are sets,

$$S \subseteq T \qquad (S \text{ is a subset of } T)$$

is a predicate equivalent to $\forall\, s : S \bullet s \in T$




$$S \subset T \qquad (S \text{ is a proper subset of } T)$$

is a predicate equivalent to $S \subseteq T \wedge S \neq T$

e.g. the following predicates are true

$\{0, 1, 2\} \subseteq \mathbb{N}$
$2 \mathinner{..} 3 \subseteq 1 \mathinner{..} 5$
$\{a, b\} \subseteq \{a, b, c\}$
$\varnothing \subseteq X$ for any set $X$
$\{x\} \subseteq X \quad \Leftrightarrow \quad x \in X$

## Power Set ($\mathbb{P}$)

If $X$ is a set,

$$\mathbb{P}\,X \qquad \text{(the power set of } X\text{)}$$

is the set of all subsets of $X$.

$$A \in \mathbb{P}\,B \quad = \quad A \subseteq B$$

e.g. the following predicates are true

$$\mathbb{P}\{a, b\} = \{\varnothing, \{a\}, \{b\}, \{a, b\}\}$$
$$\mathbb{P}\,\varnothing = \{\varnothing\} \qquad (\neq \varnothing)$$
$$1 \mathrel{..} 5 \in \mathbb{P}\,\mathbb{N}$$
$$2 \mathrel{..} 4 \in \mathbb{P}(1 \mathrel{..} 5)$$

If $X$ has $k$ elements, $\mathbb{P}\,X$ has $2^k$ elements.

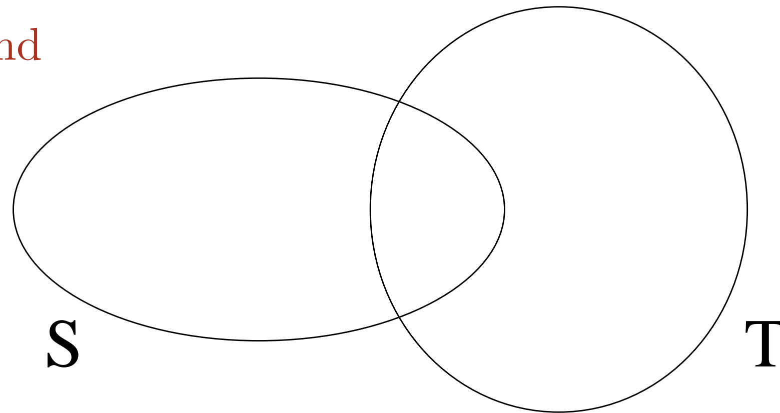
## Set Union ($\cup$)

Suppose $S, T : \mathbb{P}\, X$ (i.e. $S \subseteq X$ and $T \subseteq X$); then

$$S \cup T \qquad (S \text{ union } T)$$

is a set equal to

$$\{x : X \mid x \in S \;\vee\; x \in T\}$$




e.g. the following predicates are true

$$\{a, b, c\} \cup \{b, g, h\} = \{a, b, c, g, h\}$$
$$(1 \mathinner{..} 5) \cup (3 \mathinner{..} 7) = 1 \mathinner{..} 7$$
$$\mathbb{N}_1 \cup \{0\} = \mathbb{N}$$
$$A \cup \varnothing = A \qquad \text{(for any set } A\text{)}$$

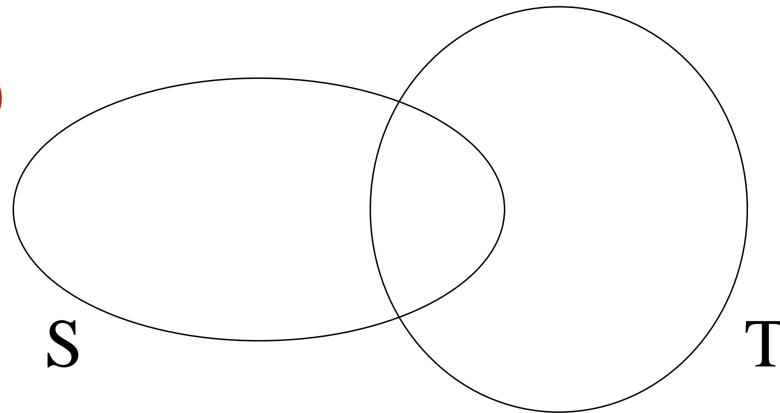
## Set Intersection ($\cap$)

Suppose $S, T : \mathbb{P}\,X$; then

$$S \cap T \qquad (S \text{ intersection } T)$$

is a set equal to

$$\{x : X \mid x \in S \;\wedge\; x \in T\}$$




e.g. the following predicates are true

$\{a, b, c\} \cap \{b, g, h\} = \{b\}$
$(1 \mathrel{..} 5) \cap (3 \mathrel{..} 7) = 3 \mathrel{..} 5$
$\{a, b, c\} \cap \{d, g\} = \varnothing$ (the sets are *disjoint*)
$A \cap \varnothing = \varnothing$ (for any set $A$)

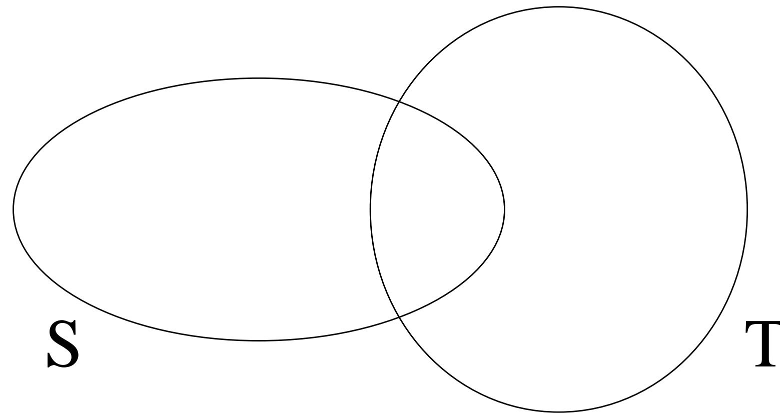
## Set Difference ($-$)

Suppose $S, T : \mathbb{P}\, X$; then

$$S - T \qquad (S \text{ subtract } T)$$

is a set equal to

$$\{x : X \mid x \in S \wedge x \notin T\}$$




e.g. the following predicates are true

$$\{a, b, c\} - \{b, g, h\} = \{a, c\}$$
$$(1 \mathinner{..} 5) - (3 \mathinner{..} 7) = 1 \mathinner{..} 2$$
$$\mathbb{N}_1 = \mathbb{N} - \{0\}$$
$$A - \varnothing = A \qquad (\text{for any set } A)$$

## Cartesian Product ($\times$)

If $A$ and $B$ are sets,

$$A \times B \qquad (A \text{ cross } B)$$

is the set of all ordered pairs $(a, b)$ with $a \in A$ and $b \in B$.

e.g. the following predicates are true

$$\{a, b\} \times \{a, c\} = \{(a, a), (a, c), (b, a), (b, c)\}$$
$$(5, -1) \in \mathbb{N} \times \mathbb{Z}$$
$$(5, -1) \notin \mathbb{N} \times \mathbb{N}$$
$$6 \notin \mathbb{N} \times \mathbb{N}$$
$$A \times \varnothing = \varnothing \qquad (\text{for any set } A)$$

$\mathbb{R} \times \mathbb{R}$ is the Cartesian plane

## Cardinality

If $X$ is any finite set,

$$\# X$$

is a natural number denoting the cardinality of (i.e. the number of elements in) $X$.

e.g.

$$\#\{a, b, c\} = 3$$

$$\#\varnothing = 0$$

$$\# \mathbb{P}\, A = 2^{\# A} \qquad \text{(for any finite set } A\text{)}$$

## Types

Z is strongly typed: every expression is given a type.

Any set can be used as a type.

The following are equivalent within set comprehension

$$(x, y) : A \times B$$
$$x : A;\ y : B$$
$$x, y : A \qquad \text{(when } B = A\text{)}$$

Notice that

$$\forall S : \mathbb{P}\, A \bullet \ldots \quad \text{not} \quad \forall S \subseteq A \bullet \ldots$$

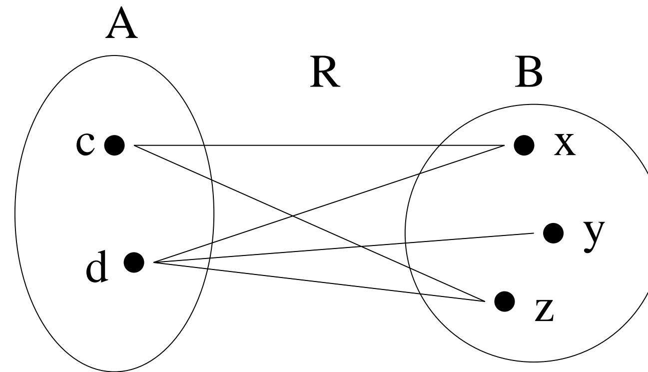
$$\forall S : \mathbb{P}\, A \bullet (\forall y : S \bullet \ldots) \quad \text{not} \quad \forall S : \mathbb{P}\, A;\ y : S \bullet \ldots$$

# Relations

A relation $R$ from $A$ to $B$, denoted by

$$R : A \leftrightarrow B,$$

is a subset of $A \times B$.




$R$ is the set $\{(c, x), (c, z), (d, x), (d, y), (d, z)\}$

**Notation:** the predicates

$$(c, z) \in R \quad \text{and} \quad c \mapsto z \in R \quad \text{and} \quad c \,\underline{R}\, z$$

are equivalent.

dom $R$ is the set $\{a : A \mid \exists\, b : B \bullet a \,\underline{R}\, b\}$

ran $R$ is the set $\{b : B \mid \exists\, a : A \bullet a \,\underline{R}\, b\}$

## Examples

$$
\begin{array}{|l}
_ \leqslant _ : \mathbb{N} \leftrightarrow \mathbb{N} \\
\hline
\forall\, x, y : \mathbb{N} \bullet \\
\qquad x \leqslant y \;\Leftrightarrow\; \exists\, k : \mathbb{N} \bullet x + k = y
\end{array}
$$

i.e. the relation $\leqslant$ is the infinite subset

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

of ordered pairs in $\mathbb{N} \times \mathbb{N}$.

$$
\begin{array}{|l}
\mathit{divides} : \mathbb{N}_1 \leftrightarrow \mathbb{N} \\
\hline
\forall\, x : \mathbb{N}_1;\; y : \mathbb{N} \bullet \\
\qquad x \,\underline{\mathit{divides}}\, y \;\Leftrightarrow\; \exists\, k : \mathbb{N} \bullet x\, k = y
\end{array}
$$

$$3 \,\underline{\mathit{divides}}\, 6 \quad \text{but} \quad \neg\, (3 \,\underline{\mathit{divides}}\, 7)$$

## Domain and Range Restriction

Suppose $R : A \leftrightarrow B$ and $S \subseteq A$ and $T \subseteq B$; then

$$S \lhd R \quad \text{is the set} \quad \{(a, b) : R \mid a \in S\}$$
$$R \rhd T \quad \text{is the set} \quad \{(a, b) : R \mid b \in T\}$$

Notice that both are true:

$$S \lhd R \in A \leftrightarrow B \qquad \text{and} \qquad R \rhd T \in A \leftrightarrow B$$

e.g. if

$$\mathit{has_sibling} : \mathit{People} \leftrightarrow \mathit{People} \quad \text{then}$$

$$\mathit{female} \lhd \mathit{has_sibling} \quad \text{is the relation} \quad \mathit{is_sister_of}$$
$$\mathit{has_sibling} \rhd \mathit{female} \quad \text{is the relation} \quad \mathit{has_sister}$$

## Domain and Range Subtraction

Suppose $R : A \leftrightarrow B$ and $S \subseteq A$ and $T \subseteq B$; then

| | | |
|---|---|---|
| $S \mathbin{⩤} R$ | is the set | $\{(a,b) : R \mid a \notin S\}$ |
| $R \mathbin{⩥} T$ | is the set | $\{(a,b) : R \mid b \notin T\}$ |

The following predicates are true

$$S \mathbin{⩤} R = (A - S) \lhd R$$
$$R \mathbin{⩥} T = R \rhd (B - T)$$
$$S \mathbin{⩤} R \in A \leftrightarrow B$$
$$R \mathbin{⩥} T \in A \leftrightarrow B$$

| | | |
|---|---|---|
| $female \mathbin{⩤} has_sibling$ | is the relation | $is_brother_of$ |
| $has_sibling \mathbin{⩥} female$ | is the relation | $has_brother$ |

## Relational Image

Suppose $R : A \leftrightarrow B$ and $S \subseteq A$

$$R(\!|\ S\ |\!) = \{b : B \mid \exists\, a : S \bullet a \underline{R}\, b\}$$

$$R(\!|\ S\ |\!) \subseteq B$$

$$\begin{aligned} ÷s(\!|\ \{8, 9\}\ |\!) \\ &\quad = \{x : \mathbb{N} \mid \exists\, k : \mathbb{N} \bullet x = 8k \ \vee \ x = 9k\} \\ &\quad = \{\text{numbers divided by 8 or 9}\} \end{aligned}$$

$$\leqslant (\!|\ \{7, 3, 21\}\ |\!) = \{x : \mathbb{N} \mid x \geqslant 3\}$$

$$has_sibling(\!|\ male\ |\!) = \{\text{people who have a brother}\}$$

## Inverse

Suppose $R : A \leftrightarrow B$

$$R^{-1} = \{(b, a) : B \times A \mid a \underline{R} b\}$$

$$R^{-1} \in B \leftrightarrow A$$

$$has_sibling^{-1} = has_sibling$$
$$divides^{-1} = has_divisor$$

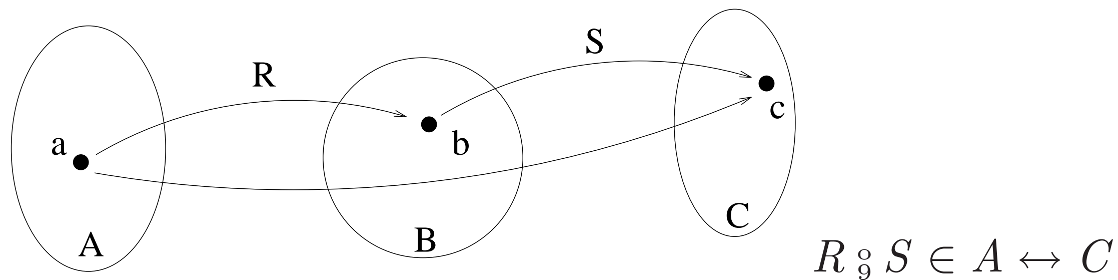
$$\begin{array}{|l} succ : \mathbb{N} \leftrightarrow \mathbb{N} \\ \hline \forall x, y : \mathbb{N} \bullet \\ \qquad x \, \underline{succ} \, y \;\Leftrightarrow\; x + 1 = y \end{array}$$

$$succ^{-1} = pred$$

## Relational Composition

Suppose $R : A \leftrightarrow B$ and $S : B \leftrightarrow C$

$$R \mathbin{;} S$$
$$= \{(a, c) : A \times C \mid \exists\, b : B \bullet a \,\underline{R}\, b \;\wedge\; b \,\underline{S}\, c\}$$




$$R \mathbin{;} S \in A \leftrightarrow C$$

e.g.

$$is_parent_of \mathbin{;} is_parent_of = is_grandparent_of$$

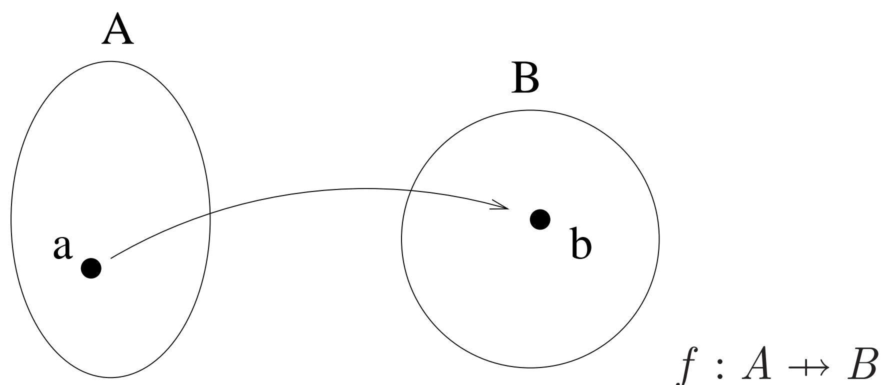
$$R^0 = \mathrm{id}[A], \quad R^1 = R, \quad R^2 = R \mathbin{;} R, \quad R^3 = R \mathbin{;} R \mathbin{;} R, \ldots$$

# Functions

A (partial) function $f$ from a set $A$ to a set $B$, denoted by

$$f : A \nrightarrow B,$$

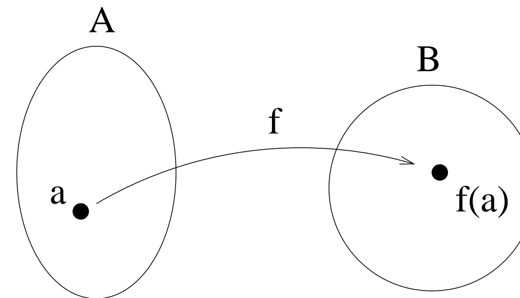
is a subset $f$ of $A \times B$ with the property that for each $a \in A$ there is at most one $b \in B$ with $(a, b) \in f$.




$\operatorname{dom} f$ is the set $\{a : A \mid \exists\, b : B \bullet (a, b) \in f\}$
$\operatorname{ran} f$ is the set $\{b : B \mid \exists\, a : A \bullet (a, b) \in f\}$

## Function Application

Suppose $f : A \nrightarrow B$ and $a \in \operatorname{dom} f$; then $f(a)$ denotes the unique *image* in $B$ that $a$ is mapped to by $f$.




The predicates

$$(a, b) \in f \quad \text{and} \quad f(a) = b$$

are equivalent.

## Total Functions

A function $f : A \nrightarrow B$ is a *total* function, denoted

$$f : A \rightarrow B,$$

if and only if $\operatorname{dom} f$ is the set $A$.

# Specifying Functions

## (1) Using a Look-up Table

If a function $f : A \nrightarrow B$ is finite (and not too large) we can specify the function explicitly by listing all the pairs $(a, b)$ in the subset of $A \times B$ where $f(a) = b$.

e.g.

$$address : PassportNo \rightarrow Address$$

| 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.

(a)

$$\begin{array}{|l} \mathit{double} : \mathbb{N} \rightarrow \mathbb{N} \\ \hline \forall\, n : \mathbb{N} \bullet \mathit{double}(n) = 2n \end{array}$$

(b)

$$\begin{array}{|l} \mathit{halve} : \mathbb{N} \nrightarrow \mathbb{N} \\ \hline \mathrm{dom}\, \mathit{halve} = \{ n : \mathbb{N} \mid \exists\, m : \mathbb{N} \bullet 2m = n \} \\ \forall\, n : \mathrm{dom}\, \mathit{halve} \bullet 2 * \mathit{halve}(n) = n \end{array}$$

(c)

$$
\begin{array}{|l}
root : \mathbb{N} \nrightarrow \mathbb{N} \\
\hline
\operatorname{dom} root = \{n : \mathbb{N} \mid \exists\, m : \mathbb{N} \bullet m^2 = n\} \\
\forall\, n : \operatorname{dom} root \bullet (root(n))^2 = n
\end{array}
$$

(d)

$$
\begin{array}{|l}
+ : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N} \\
\hline
\forall (n, m) : \mathbb{N} \times \mathbb{N} \bullet +(n, m) = n + m
\end{array}
$$

(e) Let *People* be the set of all living people.

$$
\begin{array}{|l}
birth : People \rightarrow \mathbb{N} \\
\hline
\forall\, p : People \bullet birth(p) \text{ is the year of } p\text{'s birth}
\end{array}
$$

## (3) Using Recursion

This is a variant on the previous declarative specification; a function is defined recursively in terms of itself.

e.g.

$$fact : \mathbb{N}_1 \rightarrow \mathbb{N}$$

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

so

$$fact(1) = 1$$
$$fact(2) = 2 * fact(1) = 2 * 1 = 2$$
$$fact(3) = 3 * fact(2) = 3 * 2 = 6$$
$$fact(4) = 4 * fact(3) = 4 * 6 = 24$$

and so on....

## (4) Giving an Algorithm

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

e.g.

```
input n : ℕ
var x, y: integer;
begin
x := n;  y := 0;
while x ≠ 0 do
  begin
  x := x − 1;  y := y + 2
  end;
write(y)
end.
```

This algorithm computes the function. But how can we prove this?

$$\begin{array}{|l} double : \mathbb{N} \rightarrow \mathbb{N} \\ \hline \forall\, n : \mathbb{N} \bullet double(n) = 2n \end{array}$$

## Function Overriding

Suppose $f, g : A \rightarrowtail B$; then

$$f \oplus g \quad \text{is the function} \quad (\operatorname{dom} g \lhd f) \cup g$$

i.e. the following predicates are true

$$\operatorname{dom} f \oplus g = \operatorname{dom} f \cup \operatorname{dom} g$$
$$\forall\, a : \operatorname{dom} g \bullet (f \oplus g)(a) = g(a)$$
$$\forall\, a : \operatorname{dom} f - \operatorname{dom} g \bullet (f \oplus g)(a) = f(a)$$
$$f \oplus g \in A \rightarrowtail B$$

e.g.

$$\{a \mapsto x, b \mapsto y, c \mapsto x\} \oplus \{a \mapsto y\} \;=\; \{a \mapsto y, b \mapsto y, c \mapsto x\}$$

$$\mathit{double} \oplus \mathit{root} \;=\; \{(0,0), (1,1), (2,4), (3,6), (4,2), \cdots\}$$

## Sequences

A sequence $s$ of elements from a set $A$, denoted

$$s : \mathrm{seq}\, A,$$

is a function $s : \mathbb{N} \nrightarrow A$ where $\mathrm{dom}\, s = 1 \mathinner{..} n$ for some natural number $n$. For example,

$$\langle b, a, c, b \rangle \text{ denotes the sequence (function)} \{1 \mapsto b, 2 \mapsto a, 3 \mapsto c, 4 \mapsto b\}$$

The empty sequence is denoted by $\langle\,\rangle$.

The set of all sequences of elements from $A$ is denoted $\mathrm{seq}\, A$ and is defined to be

$$\mathrm{seq}\, A == \{s : \mathbb{N} \nrightarrow A \mid \exists\, n : \mathbb{N} \bullet \mathrm{dom}\, s = 1 \mathinner{..} n\}$$

We define $\mathrm{seq}_1\, A$ to be the set of all non-empty sequences, i.e.

$$\mathrm{seq}_1\, A == \mathrm{seq}\, A - \{\langle\,\rangle\}$$

Notice that: $\langle a, b, a \rangle \neq \langle a, a, b \rangle \neq \langle a, b \rangle$

# Special Functions for Sequences

## Concatenation

$$\langle a, b \rangle ^\frown \langle b, a, c \rangle = \langle a, b, b, a, c \rangle$$

## Head

$$head : \mathrm{seq}_1\, A \rightarrow A$$

$$\forall\, s : \mathrm{seq}_1\, A \bullet head(s) = s(1)$$

$$\mathrm{head}\langle c, b, b \rangle = c$$

## Tail

$$tail : \mathrm{seq}_1\, A \rightarrow \mathrm{seq}\, A$$

$$\forall\, s : \mathrm{seq}_1\, A \bullet \langle head(s) \rangle ^\frown tail(s) = s$$

$$tail\langle c, b, b \rangle = \langle b, b \rangle$$

# Part 2 — Z Specifications

## Z Case Study: A Message Buffer

- A number of messages are transmitted from one location to another.
- Because of other traffic on the line each message for transmission is placed in a buffer which outputs the message when the line is free.
- This buffer may contain several messages at any time, but there is a fixed upper limit on the number of messages the buffer may contain.
- The buffer operates on a first in/first out (FIFO) principle.

# Formal Specification

## The State Schema

$[MSG]$ (The exact nature of these messages is not important)

is the set of all possible messages that could ever be transmitted.

$| \; max : \mathbb{N}$ (The actual value of $max$ is not important)

is the constant maximum number of messages that can be held in the buffer at any one time.

*Buffer*

$items : \text{seq}\, MSG$ — declaration

$\#items \leqslant max$ — predicate

e.g. suppose $MSG = \{m_1, m_2, m_3\}$ and $max = 4$

Then $items = \langle m_1, m_2 \rangle$ is an instance, but $items = \langle m_3, m_1, m_1, m_2, m_2 \rangle$ is not

- a schema specifies a relationship between variable values; *Buffer* is a *state* schema
- a state schema 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 is 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 *Buffer* gives a static view of the system. To specify how the system can change we need to specify *operation* schema.
An operation can be thought of 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.
We adopt the convention that the value of state variables before the operation are denoted by unprimed identifiers, while values after the operation are denoted by primed identifiers.
For the message buffer there are two operations:

*Join* (a new message is added to the buffer)

*Leave* (a message leaves the buffer)

## The Join Operation

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{Join}} \\
\hline
items, items' : \mathrm{seq}\, MSG \\
msg? : MSG \\
\hline
\#items \leqslant max \\
\#items' \leqslant max \\
\#items < max \\
items' = items \frown \langle msg? \rangle \\
\hline
\end{array}
$$

- *items* denotes the sequence of messages in the buffer before the operation
- $items'$ denotes the sequence of messages in the buffer after the operation
- the decoration ? denotes an *input*
- there is an implicit $\wedge$ between each line

- 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 already be completely full
- the last line of the predicate specifies the relationship between the buffer contents before and after the operation: the input message is appended to the sequence of messages already in the buffer. e.g. suppose

  $$MSG = \{m_1, m_2, m_3\} \quad \text{and} \quad max = 4 \quad \text{and}$$
  $$items = \langle m_1, m_2, m_1 \rangle \quad \text{and} \quad msg? = m_3;$$

  then after the operation

  $$items = \langle m_1, m_2, m_1, m_3 \rangle$$

## Schema Inclusion

Because we always have a 'before' and 'after' instance of the state schema for any operation we make the following syntactic simplification: define

$$\Delta Buffer$$

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

$$\#items \leqslant max$$
$$\#items' \leqslant max$$

and we can now write *Join* by including this schema:

$$Join$$

$$\Delta Buffer$$
$$msg? : MSG$$

$$\#items < max$$
$$items' = items \frown \langle msg? \rangle$$

In general, including a schema in the declaration part of another schema means that the included schema has its declaration added to the new schema, and its predicate conjoined to the predicate of the new schema.
e.g. if

| $A$ |
| --- |
| $x : T_1$ |
| $y : T_2$ |
| $P(x, y)$ |

| $S$ |
| --- |
| $A$ |
| $z : T_3$ |
| $Q(x, y, z)$ |

then $S$ expands to the schema

| $S$ |
| --- |
| $x : T_1$ |
| $y : T_2$ |
| $z : T_3$ |
| $P(x, y) \wedge Q(x, y, z)$ |

## The Leave Operation

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{Leave}} \\
\hline
\Delta Buffer \\
msg! : MSG \\
\hline
items \neq \varnothing \\
items = \langle msg! \rangle ^\frown items' \\
\hline
\end{array}
$$

- the decoration ! denotes an *output*

- 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

- 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 in the buffer.

## The Initial State

To complete the specification of the message buffer we need to specify the initial state of the buffer:

$$
\begin{array}{|l}
\hline
\textit{Buffer}_{\text{INIT}} \\
\textit{Buffer} \\
\hline
\textit{items} = \langle\ \rangle \\
\hline
\end{array}
$$

i.e.

$$
\begin{array}{|l}
\hline
\textit{Buffer}_{\text{INIT}} \\
\textit{items} : \operatorname{seq} \textit{MSG} \\
\hline
\#\textit{items} \leqslant \textit{max} \\
\textit{items} = \langle\ \rangle \\
\hline
\end{array}
$$

## Conclusions

We have specified the message buffer in terms of what an observer of the buffer can expect to see. Initially the buffer would be empty, and then the operations of *Join* and *Leave* can occur whenever they are enabled (i.e. when their pre-conditions are satisfied). Operations are assumed to be atomic (i.e. occur instantaneously). At all times an observer would notice that the state schema for the buffer is satisfied.

# Extending Specifications

## Example: A Slow Buffer

$$delay : \mathbb{N}$$

**SlowBuffer**

$$Buffer$$
$$idle : \mathbb{N}$$

i.e.

**SlowBuffer**

$$items : \text{seq}\ MSG;\ \ idle : \mathbb{N}$$

---

$$\#items \leqslant max$$

**SlowBuffer$_{\text{INIT}}$**

$$SlowBuffer$$
$$Buffer_{\text{INIT}}$$

---

$$idle = 0$$

i.e.

**SlowBuffer$_{\text{INIT}}$**

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

---

$$\#items \leqslant max \land items = \langle\ \rangle \land idle = 0$$

## Merging Schemas

| $A$ |
| --- |
| $x : T_1$ |
| $y : T_2$ |
| $P(x, y)$ |

| $B$ |
| --- |
| $y : T_2$ |
| $z : T_3$ |
| $Q(y, z)$ |

| $C$ |
| --- |
| $A$ |
| $B$ |

where

| $C$ |
| --- |
| $x : T_1$ |
| $y : T_2$ |
| $z : T_3$ |
| $P(x, y) \wedge Q(y, z)$ |

- type compatibility is needed to merge schemas

## Slow Operations

$$\begin{array}{|l} \multicolumn{1}{l}{\textit{SlowJoin}} \\ \Delta \textit{SlowBuffer} \\ \textit{Join} \\ \hline \textit{idle} \geqslant \textit{delay} \\ \textit{idle}' = 0 \\ \hline \end{array}$$

i.e.

$$\begin{array}{|l} \multicolumn{1}{l}{\textit{SlowJoin}} \\ \textit{items}, \textit{items}' : \operatorname{seq} \textit{MSG} \\ \textit{idle}, \textit{idle}' : \mathbb{N} \\ \textit{msg}? : \textit{MSG} \\ \hline \#\textit{items} \leqslant \textit{max} \land \#\textit{items}' \leqslant \textit{max} \\ \#\textit{items} < \textit{max} \\ \textit{items}' = \textit{items} \frown \langle \textit{msg}? \rangle \\ \textit{idle} \geqslant \textit{delay} \land \textit{idle}' = 0 \\ \hline \end{array}$$

**SlowLeave**

$$\Delta SlowBuffer$$
$$Leave$$

---

$$idle \geqslant delay \wedge idle' = 0$$

**Tick**

$$\Delta SlowBuffer$$

---

$$idle' = idle + 1 \wedge items' = items$$

Exercise:
give the expanded form of the operation schemas *SlowLeave* and *Tick*.

## Reasoning About the Specification

Can we verify that the message buffer as specified has the FIFO property, i.e. messages leave the buffer in the same order as they arrive?
To do this we introduce *auxiliary* variables which do not alter the functionality of the specification but aid in the analysis.
In this case we introduce auxiliary sequences *inhist* and *outhist* to record the history of the flow of messages into and out of the buffer.
The new system obtained from the buffer by adding these auxiliary variables can be specified by including the original schemas into new schemas which contain the extra information about auxiliary variables.

**RecordedBuffer**

$$Buffer$$
$$inhist : \mathrm{seq}\, MSG$$
$$outhist : \mathrm{seq}\, MSG$$

---

**RecordedBuffer_INIT**

$$RecordedBuffer$$
$$Buffer_{\textsc{Init}}$$

$$inhist = \langle\ \rangle$$
$$outhist = \langle\ \rangle$$

---

**RecordedJoin**

$$\Delta RecordedBuffer$$
$$Join$$

$$inhist' = inhist \frown \langle msg? \rangle$$
$$outhist' = outhist$$

---

**RecordedLeave**

$$\Delta RecordedBuffer$$
$$Leave$$

$$inhist' = inhist$$
$$outhist' = outhist \frown \langle msg! \rangle$$

e.g. the schema *RecordedJoin* with the included schemas expanded becomes:

**RecordedJoin**

$$
\begin{array}{l}
items, items' : \mathrm{seq}\, MSG \\
inhist, inhist' : \mathrm{seq}\, MSG \\
outhist, outhist' : \mathrm{seq}\, MSG \\
msg? : MSG \\
\hline
\#items \leqslant max \\
\#items' \leqslant max \\
\#items < max \\
items' = items \frown \langle msg? \rangle \\
inhist' = inhist \frown \langle msg? \rangle \\
outhist' = outhist
\end{array}
$$

How can we use the auxiliary variables to prove that the buffer satisfies the FIFO property ?

**Theorem**

$$\forall \, RecordedBuffer \bullet inhist = outhist \frown items$$

*Proof:*
Use structural induction.
Initially $inhist = outhist = items = \langle \, \rangle$,
so the predicate is true.

Suppose the predicate is true, and *RecordedJoin* occurs.
After the operation

$$inhist' = inhist \frown \langle msg? \rangle \land outhist' = outhist \land items' = items \frown \langle msg? \rangle$$

Hence: $inhist'$

$$= inhist \frown \langle msg? \rangle = (outhist \frown items) \frown \langle msg? \rangle$$
$$= outhist \frown (items \frown \langle msg? \rangle) = outhist' \frown items'$$

and the predicate remains true. A similar argument shows that the operation *RecordedLeave* also preserves the predicate. $\Box$

## Conjunction

$$SlowRecordedBuffer \mathrel{\widehat=} SlowBuffer \land RecordedBuffer$$

is equivalent to merging the schemas:

Also

$$SlowRecordedBuffer_{\text{INIT}} \mathrel{\widehat=} SlowBuffer_{\text{INIT}} \land RecordedBuffer_{\text{INIT}}$$

$$SlowRecordedJoin \mathrel{\widehat=} SlowJoin \land RecordedJoin$$

If $A$ and $B$ are schemas:

- the declaration of $A \land B$ is the union of the declarations of $A$ and $B$;
- the predicate of $A \land B$ is the conjunction of the predicates of $A$ and $B$.

## Disjunction

$$Flag ::= ok \mid error$$

**JoinOK**

$$Join$$
$$flag! : Flag$$

---

$$flag! = ok$$

**JoinError**

$$\Xi Buffer$$
$$flag! : Flag$$

---

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

$$CompleteJoin \mathrel{\widehat=} JoinOK \lor JoinError$$

**CompleteJoin**

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

---

$$\#items < max \land items' = items \frown \langle msg? \rangle \land flag! = ok$$
$$\lor$$
$$\#items = max \land items' = items \land flag! = error$$

If $A$ 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 $B$.

In general

$A$

$x : T_1$
$y : T_2$

$P(x, y)$

$B$

$y : T_2$
$z : T_3$

$Q(y, z)$

**conjunction**

$A \wedge B$

$x : T_1;\ y : T_2;\ z : T_3$

$P(x, y) \wedge Q(y, z)$

**disjunction**

$A \vee B$

$x : T_1;\ y : T_2;\ z : T_3$

$P(x, y) \vee Q(y, z)$

## Composition

$$Join \mathbin{;} Leave$$

is an (atomic) operation with the effect of a *Join* followed by a *Leave*. Defining $JoinLeave \mathrel{\widehat=} Join \mathbin{;} Leave$ gives

**JoinLeave**

$\Delta Buffer$
$msg?, msg! : MSG$

---

$\# items < max$
$\exists\, items'' : \mathrm{seq}\, MSG \bullet items'' = items \frown \langle msg? \rangle \land items'' = \langle msg! \rangle \frown items'$

- the pre-state of *Join* is the pre-state of $Join \mathbin{;} Leave$
- the post-state of *Join* is identified with the pre-state of *Leave* and hidden within $Join \mathbin{;} Leave$
- the consequent post-state of *Leave* is the post-state of $Join \mathbin{;} Leave$

## Composition in general

**A**

$$x : T_1$$
$$y : T_2$$

---

$$P(x, y)$$

**$AOP_1$**

$$\Delta A$$
$$t_3? : T_3$$
$$t_4! : T_4$$

---

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

**$AOP_2$**

$$\Delta A$$
$$t_5? : T_5$$
$$t_6! : T_6$$

---

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

**$AOP_1 \fatsemi AOP_2$**

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

---

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

# Piping

$$\begin{array}{|l}
\hline \textit{Duplicate} \\
msg? : MSG;\ duplicate! : \mathrm{seq}\, MSG \\
\hline
duplicate! = \langle msg?, msg? \rangle \\
\hline
\end{array}$$

$$LeaveDuplicated \mathrel{\widehat=} Leave \gg Duplicate$$

$$\begin{array}{|l}
\hline \textit{LeaveDuplicated} \\
\Delta Buffer \\
duplicate! : \mathrm{seq}\, MSG \\
\hline
items \neq \langle\, \rangle \\
\exists\, m : MSG \bullet items = \langle m \rangle \frown items' \land duplicate! = \langle m, m \rangle \\
\hline
\end{array}$$

- the output variables of *Leave* and the input variables of *Duplicate* with identical bases (i.e. ignoring the decorations '?' and '!' respectively) have their values identified and hidden in $Leave \gg Duplicate$.

**Piping in general**

$$
\begin{array}{|l}
\hline
\textbf{Schema } A \\
x : T_1;\ y : T_2 \\
\hline
P(x, y) \\
\hline
\end{array}
\qquad
\begin{array}{|l}
\hline
\textbf{Schema } D \\
v : T_3;\ w : T_4 \\
\hline
Q(v, w) \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
\textbf{Schema } AOP \\
\Delta A \\
t_5? : T_5 \\
t_6! : T_6 \\
\hline
RA(x, x', y, y', t_5?, t_6!) \\
\hline
\end{array}
\qquad
\begin{array}{|l}
\hline
\textbf{Schema } DOP \\
\Delta D \\
t_6? : T_6 \\
t_7! : T_7 \\
\hline
RD(v, v', w, w', t_6?, t_7!) \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
\textbf{Schema } AOP \gg DOP \\
\Delta A \\
\Delta D \\
t_5? : T_5 \\
t_7! : T_7 \\
\hline
\exists\, t : T_6 \bullet RA(x, x', y, y', t_5?, t) \wedge RD(v, v', w, w', t, t_7!) \\
\hline
\end{array}
$$

## Non-determinism

$$\begin{array}{l} G \\ \hline a, b, c : \mathbb{N} \\ \hline a \leqslant b \wedge a = c \end{array}$$

$$\begin{array}{l} GOP1 \\ \hline \Delta G \\ \hline a' = 3 \end{array}$$

- $b'$ can take any value $\geqslant 3$ regardless of the value of $b$
- $c' = 3$ because the state invariant gives $a' = c'$
- if the value of $b$ is to remain unchanged, $b = b'$ must be added

$$\begin{array}{l} GOP2 \\ \hline \Delta G \\ out! : \mathbb{N} \\ \hline out! = a + b + c \end{array}$$

- the values of $a'$, $b'$, $c'$ are undetermined except that $a' \leqslant b'$ and $a' = c'$
- compare with using $\Xi G$

## Renaming

$$\begin{array}{|l}\hline A \\ x : T_1 \\ y : T_2 \\ \hline P(x, y) \\ \hline\end{array}$$

$R \mathrel{\widehat{=}} A[z/y]$ expands to

$$\begin{array}{|l}\hline R \\ x : T_1 \\ z : T_2 \\ \hline P(x, z) \\ \hline\end{array}$$

e.g $\quad TwoJoins \mathrel{\widehat{=}} Join[msg_1?/msg?] \mathbin{⨾} Join[msg_2?/msg?]$

$$\begin{array}{|l}\hline TwoJoins \\ \Delta Buffer \\ msg_1?, msg_2? : MSG \\ \hline \# items < max - 1 \land items' = items \frown \langle msg_1? \rangle \frown \langle msg_2? \rangle \\ \hline\end{array}$$

(compare this with $Join \mathbin{⨾} Join$)

# Schemas as Types

## Instantiation

- a schema determines a type
- a variable of type schema (an instance) can be declared
- variables within an instance are referenced using the 'dot' notation

e.g.

$$\begin{array}{l} \textit{TwoBuffers} \\ \hline a, b : \textit{Buffer} \\ \hline a.items = b.items \end{array}$$

expands to give

$$\begin{array}{l} \textit{TwoBuffers} \\ \hline a, b : \textit{Buffer} \\ \hline \#a.items \leqslant max \\ \#b.items \leqslant max \\ a.items = b.items \end{array}$$

In general, if

$A$

$x : T_1$
$y : T_2$

---

$P(x, y)$

then

$I$

$a : A$

expands to give

$I$

$a : A$

---

$P(a.x, a.y)$

$H1$

$A$
$x : T_3$

has a type clash unless $T_1 = T_3$

$H2$

$a : A$
$x : T_3$

has no type clash

## Global Definitions (constants)

$$\begin{array}{|l} square : \mathbb{N} \rightarrow \mathbb{N} \\ \hline \forall\, n : \mathbb{N} \bullet square(n) = n^2 \end{array}$$

$$\begin{array}{|l} temp : \mathbb{N} \\ \hline temp < 451 \end{array}$$

$$\begin{array}{|l} max : \mathbb{N} \end{array}$$

Schemas can be used as types in global definition:

$$\begin{array}{|l} length : Buffer \rightarrow \mathbb{N} \\ \hline \forall\, b : Buffer \bullet \\ \quad\quad length(b) = \#b.items \end{array}$$

# Generic Typing

**In schemas:**

$$\textit{Buffer}[T]$$
$$items : \mathrm{seq}\ T$$
$$\#items \leqslant max$$

$$\textit{Join}[T]$$
$$\Delta Buffer[T]$$
$$t? : T$$
$$\#items < max \wedge items' = items \frown \langle t? \rangle$$

**In global definitions:**

$$[T]$$
$$\mathrm{head} : \mathrm{seq}_1\ T \rightarrow T$$
$$\forall\, s : \mathrm{seq}_1\ T \bullet \mathrm{head}\ s = s(1)$$

## Alternative Syntax

$$\begin{array}{|l}\hline \multicolumn{1}{l}{A} \\ x : T_1 \\ y : T_2 \\ \hline P(x, y) \\ \hline \end{array}$$

can be written as

$$A \mathrel{\widehat=} [x : T_1;\ y : T_2 \mid P(x, y)]$$

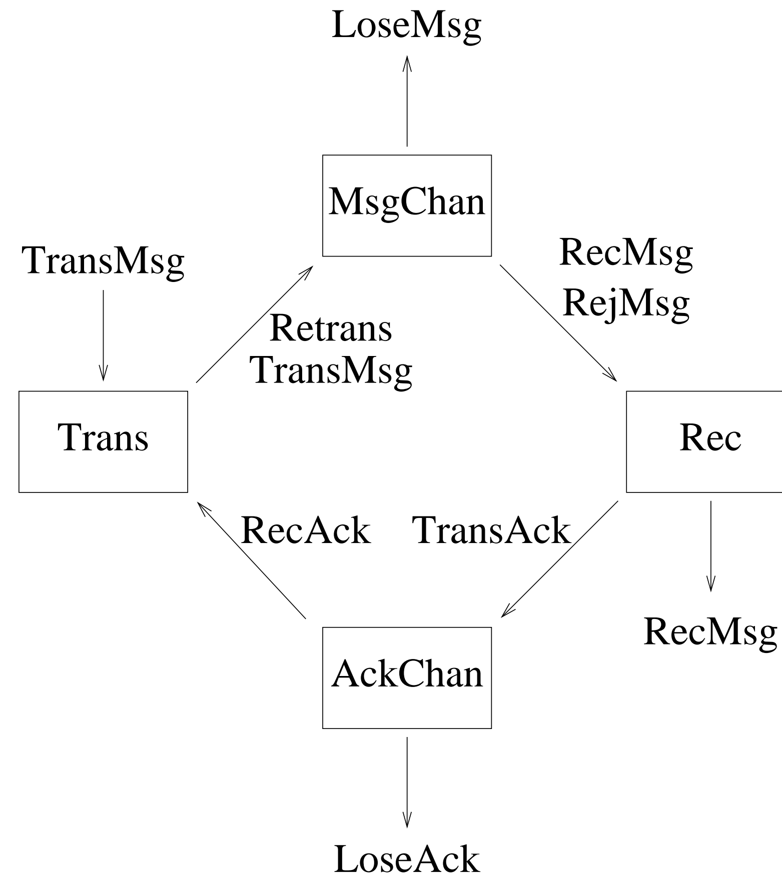
e.g.

$$\mathit{Buffer} \mathrel{\widehat=} [\mathit{items} : \operatorname{seq} \mathit{MSG} \mid \#\mathit{items} \leqslant \mathit{max}]$$

$$\mathit{SlowBuffer}_{\mathrm{INIT}} \mathrel{\widehat=} [\mathit{SlowBuffer};\ \mathit{Buffer}_{\mathrm{INIT}} \mid \mathit{idle} = 0]$$

$$\Delta \mathit{Buffer} \mathrel{\widehat=} [\mathit{Buffer};\ \mathit{Buffer}'] \qquad (= \mathit{Buffer} \land \mathit{Buffer}')$$

# Z Case Study: Alternating-Bit Protocol



$$Tag == \{0, 1\}$$

$$[MSG]$$

$$TagMsg == Tag \times MSG$$

**Trans**

$$
\begin{array}{l}
buf : \mathrm{seq}\ TagMsg \\
tag : Tag \\
\hline
\# buf \leqslant 1 \\
\forall t : Tag;\ m : MSG \bullet \\
\quad buf = \langle (t, m) \rangle \Rightarrow tag = t
\end{array}
$$

**Rec**

$$exptag : Tag$$

**MsgChan**

$$msgchan : \mathrm{seq}\ TagMsg$$

**AckChan**

$$ackchan : \mathrm{seq}\ Tag$$

$$State \mathrel{\widehat=} Trans \land Rec \land MsgChan \land AckChan$$

- *buf* contains any tagged message that has been transmitted but not yet acknowledged
- *tag* is the tag of the last tagged message to be transmitted
- *exptag* is the tag of the next message expected by the receiver
- *msgchan* is the sequence of tagged messages on route to the receiver
- *ackchan* is the sequence of tags of messages acknowledged by the receiver on route to the transmitter

$State_{\textsc{Init}}$

$State$

---

$buf = \langle\ \rangle$
$tag = 0$
$msgchan = \langle\ \rangle$
$exptag = 1$
$ackchan = \langle\ \rangle$

## Operations

If *buf* is empty, a message can be accepted from the environment, tagged and this tagged message transmitted.

$$
\begin{array}{|l}
\text{_ } \mathit{TransMsg} \text{ ____________________} \\
\Delta \mathit{Trans} \\
\Delta \mathit{MsgChan} \\
m? : \mathit{MSG} \\
\hline
\mathit{buf} = \langle\, \rangle \\
\mathit{tag}' = 1 - \mathit{tag} \\
\mathit{buf}' = \langle (\mathit{tag}', m?) \rangle \\
\mathit{msgchan}' = \mathit{msgchan} \frown \mathit{buf}' \\
\hline
\end{array}
$$

If *buf* is not empty, its contents can be re-transmitted.

$$
\begin{array}{|l}
\text{_ } \mathit{Retrans} \text{ ____________________} \\
\Xi \mathit{Trans} \\
\Delta \mathit{MsgChan} \\
\hline
\mathit{buf} \neq \langle\, \rangle \\
\mathit{msgchan}' = \mathit{msgchan} \frown \mathit{buf} \\
\hline
\end{array}
$$

If *msgchan* is not empty, the tagged message at its head can be accepted by the receiver.

If its tag is the expected tag, the message is output to the environment.

$$
\begin{array}{|l}
\textit{RecMsg} \\
\hline
\Delta \textit{MsgChan} \\
\Delta \textit{Rec} \\
m! : \textit{MSG} \\
\hline
\textit{msgchan} \neq \langle\ \rangle \\
(\textit{exptag}, m!) = \mathrm{head}\ \textit{msgchan} \\
\textit{msgchan}' = \mathrm{tail}\ \textit{msgchan} \\
\textit{exptag}' = 1 - \textit{exptag} \\
\hline
\end{array}
$$

If the tag of the tagged message at the head of *msgchan* is not the expected tag, the message is rejected.

$$
\begin{array}{|l}
\textit{RejMsg} \\
\hline
\Delta \textit{MsgChan} \\
\Xi \textit{Rec} \\
\hline
\textit{msgchan} \neq \langle\ \rangle \\
\nexists m : \textit{MSG} \bullet \\
\quad (\textit{exptag}, m) = \mathrm{head}\ \textit{msgchan} \\
\textit{msgchan}' = \mathrm{tail}\ \textit{msgchan} \\
\hline
\end{array}
$$

The tag of the last message output to the environment can be transmitted back as an acknowledgement.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{TransAck}} \\
\hline
\Xi Rec \\
\Delta Ackchan \\
\hline
ackchan' = ackchan \frown \langle 1 - exptag \rangle \\
\hline
\end{array}
$$

When an acknowledgement is received by the transmitter, *buf* is emptied if the acknowledgement equals *tag*; otherwise it is rejected.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{RecAck}} \\
\hline
\Delta AckChan \\
\Delta Trans \\
\hline
ackchan \neq \langle\ \rangle \\
ackchan' = \mathrm{tail}\ ackchan \\
tag = \mathrm{head}\ ackchan \;\Rightarrow\; buf' = \langle\ \rangle \\
tag \neq \mathrm{head}\ ackchan \;\Rightarrow\; buf' = buf \\
tag' = tag \\
\hline
\end{array}
$$

At any time the tagged message at the head of *msgchan* can be lost.

$$
\begin{array}{l}
\textit{LoseMsg} \\
\hline
\Delta \textit{MsgChan} \\
\hline
\textit{msgchan} \neq \langle \, \rangle \\
\textit{msgchan}' = \mathrm{tail}\, \textit{msgchan}
\end{array}
$$

At any time the acknowledgement at the head of *ackchan* can be lost.

$$
\begin{array}{l}
\textit{LoseAck} \\
\hline
\Delta \textit{AckChan} \\
\hline
\textit{ackchan} \neq \langle \, \rangle \\
\textit{ackchan}' = \mathrm{tail}\, \textit{ackchan}
\end{array}
$$

## Behavioural Modelling in Z

- move from an initial state to successor states by a sequence of enabled operations
- no inbuilt constraints on the selection of enabled operations
- history (trace) constraints must be explicitly introduced with history variables
- non-determinism in both operation specification and in the selection of enabled operations