U O W

Software Requirements, Specifications and Formal Methods

A/Prof. Lei Niu

UNIVERSITY
OF WOLLONGONG
AUSTRALIA

Using Z in requirement specification in real world system

This lecture will show examples on how to use Z in requirement specification in some real-world systems, such as

- Document control system
- Text processing
- Eight queens problem
- An automated billing system

Document control system

A simple document control system allows people work together and share their work. But it *may cause errors when two people are working on the same file*. We can enlist the computer to help prevent such errors. Here is an excerpt from the informal description:

- If a user wants to *check out a document* in order *to change the document* and the user has the permission to change it, and *nobody else is changing it at the moment*, then that user may check the document out.
- As soon as *a user has checked out a document for editing, everyone else is disallowed from checking it out* (of course people with read permission can read it).
- When *the user is done editing the document*, *it should be checked in, allowing another user to check it out*.

UNIVERSITY
OF WOLLONGONG
AUSTRALIA

Document control system

- What are the data types involved?
  - (Define the data type)
- What are the schemas involved?
  - (Define the system state schema)
- What are the operations involved?
  - (Create full operation schemas)
    - successful scenarios
    - non-successful scenarios
    - combine successful and non-successful scenarios

Document control system

- We start with the definition of basic types:
  - [PERSON, DOCUMENT]

- Some people have permission to change or read particular documents. We can model that as a relation

$$permission : DOCUMENT \leftrightarrow PERSON$$

$$doug, aki, phil : PERSON$$
$$spec, design, code : DOCUMENT$$

$$\begin{aligned} permission = \{&(spec, doug), (design, doug), (design, aki), (code, aki), \\ &(code, phil)\} \end{aligned}$$

Document control system

- A document can only be checked out to one person at a time, so it is an *<u>injection</u>* which *<u>associates each document with a single person</u>*. So we define another *checked_out relation as an injection*, which is a subset of permission

$$\begin{array}{|l}
\textit{Documents} \\
\hline
checked_out : DOCUMENT \rightarrowtail\!\!\!\!\!+ PERSON \\
\hline
checked_out \subseteq permission \\
\hline
\end{array}$$

- checked_out is a partial function, which indicate that documents can only be checked out to people who have permission to change them

$$checked\_out = \{(design, doug), (spec, doug), (code, phil)\}$$

Document control system

- Two operations shall be defined to change the state of the documents, i.e., CheckOut and CheckIn

CheckOut

$$\Delta Documents$$
$$p? : PERSON$$
$$d? : DOCUMENT$$

$$d? \notin \mathrm{dom}\ checked\_out$$
$$(d?, p?) \in permission$$
$$checked\_out' = checked\_out \cup \{(d?, p?)\}$$

Document d has not been checked out yet

Document control system

- CheckOut has two preconditions. We need to consider the exceptional cases

- When the document was *already checked out,*

$$CheckedOut \mathrel{\widehat=} [\, \Xi Documents;\ d? : DOCUMENT \mid d? \in \operatorname{dom} checked\_out \,]$$

- When the person *does not have the permission,*

$$
\begin{array}{|l}
\textbf{Unauthorized} \\
\hline
\Xi Documents \\
p? : PERSON \\
d? : DOCUMENT \\
\hline
(d?, p?) \notin permission \\
\hline
\end{array}
$$

- So, the total operation T_CheckOut is

$$T\_CheckOut \mathrel{\widehat=} CheckOut \lor CheckedOut \lor Unauthorized$$

Document control system

- Then we can define the T_CheckIn as follows

$$
\begin{array}{|l}
\textbf{CheckIn} \\
\hline
\Delta Documents \\
d?: DOCUMENT \\
\hline
d? \in \mathrm{dom}\ checked\_out \\
checked\_out' = \{d?\} \ntriangleleft checked\_out \\
\hline
\end{array}
$$

$$CheckedIn \mathrel{\widehat=} [\Xi Documents; d?: DOCUMENT \mid d? \notin \mathrm{dom}\ checked\_out]$$

$$T\_CheckIn \mathrel{\widehat=} CheckIn \lor CheckedIn$$

Text processing

- We continue the simple text editor by defining more characters:

- TEXT includes the empty sequence, but *SPACE and WORD must have at least one character*. *LINE is a special blank character which just breaks a line*.

- Declaration: SPACE, TEXT, WORD, LINE, etc.?

Text processing

- We continue the simple text editor by defining more characters:

$$[CHAR]$$

$$\mid \; blank : \mathbb{P}\, CHAR$$

empty spaces

$$TEXT == \operatorname{seq} CHAR$$

$$SPACE == \operatorname{seq}_1 blank$$

$$WORD == \operatorname{seq}_1 (CHAR \setminus blank)$$

$$LINE \in blank$$

- TEXT includes the empty sequence, but *SPACE and WORD must have at least one character*. *LINE is a special blank character which just breaks a line*.

Text processing

- Then we define a *total function* words to divide TEXT to a sequence of WORD (for word accounting purpose)

$$
\begin{array}{|l}
words : TEXT \rightarrow \operatorname{seq} WORD \\
\hline
\forall s : SPACE;\ w : WORD;\ l, r : TEXT \bullet \\
\quad words\ \langle\rangle = \langle\rangle\ \wedge \\
\quad words\ s = \langle\rangle\ \wedge \\
\quad words\ w = \langle w \rangle\ \wedge \\
\quad words\ (s \frown r) = words\ r\ \wedge \\
\quad words\ (l \frown s) = words\ l\ \wedge \\
\quad words\ (l \frown s \frown r) = (words\ l) \frown (words\ r)
\end{array}
$$

$$words\ \langle H, o, w, \ , a, r, e, \ , y, o, u, ? \rangle = \langle \langle H, o, w \rangle, \langle a, r, e \rangle, \langle y, o, u \rangle \rangle$$

- #(words t) will return the number of words in the TEXT t

Text processing

- Filling paragraphs

We can define a *fill operation* to transforms raggedy-looking text with lines of different lengths into nicely formatted text with lines nearly the same length. For example

```
Almost any text editor provides a fill
operation. The fill operation transforms raggedy-looking text
with lines of
different lengths into nicely formatted text with lines
nearly the same length.
```

shall be transformed to

```
Almost any text editor provides a fill operation.  The
fill operation transforms raggedy-looking text with lines of
different lengths into nicely formatted text with lines
nearly the same length.
```

Text processing

- The *fill operation* can be considered as a special case of the format operation that changes the appearance of a text by *breaking lines in different places* and *expanding or contracting the spaces between words*, subject to the constraint that *no line exceeds the page width*.
- The *operation must not change the content of the text*.

$width : \mathbb{N}$

$lines : TEXT \rightarrow \operatorname{seq} LINE$

... definition omitted ...

Format

$t, t' : TEXT$

$words\ t' = words\ t$

$\forall l : \operatorname{ran} (lines\ t') \bullet \#l \leq width$

Text words cannot be changed

Fill

$Format$

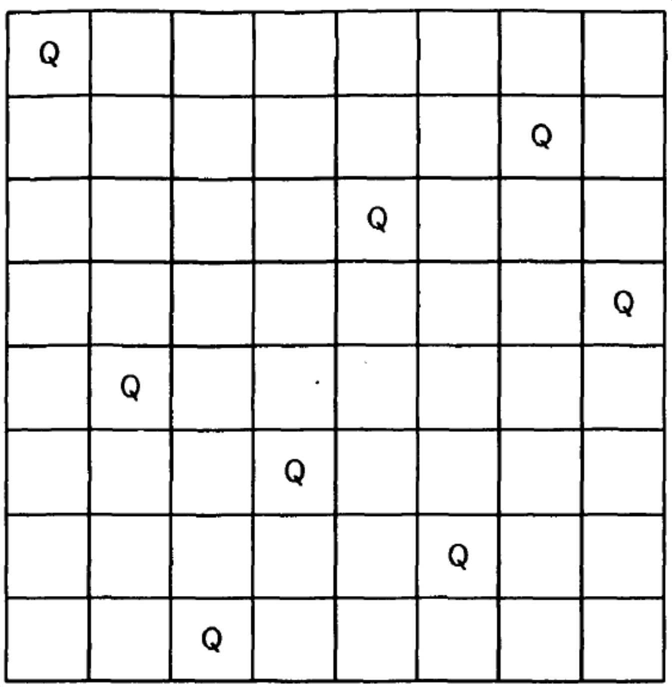
$\#(lines\ t') = min\ \{t' : TEXT \mid Format \bullet \#(lines\ t')\}$

Eight queens

Problem: Eight queens must be placed on a chessboard so that no queen attacks any others. A chessboard is a square grid with eight columns, or files, and eight rows, or ranks. *__When a queen is placed on a square, it attacks any other queen that sits on the same rank, file, or diagonals.__*

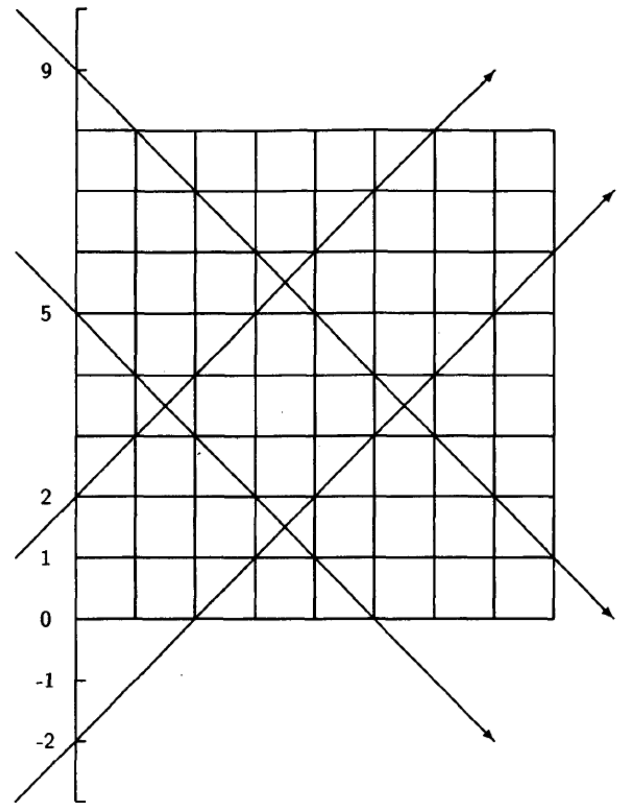
1 2 3 4 5 6 7 8 (FILE)

8 7 6 5 4 3 2 1



(RANK)

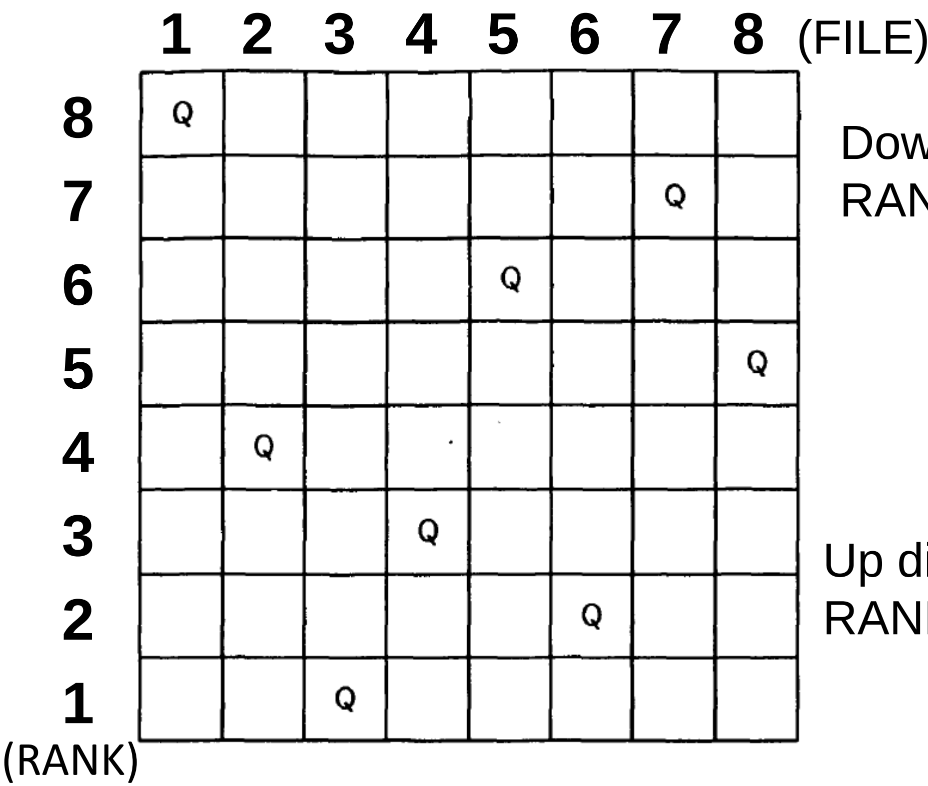
Declarations:
FILE
RANK ?
DIAGONAL

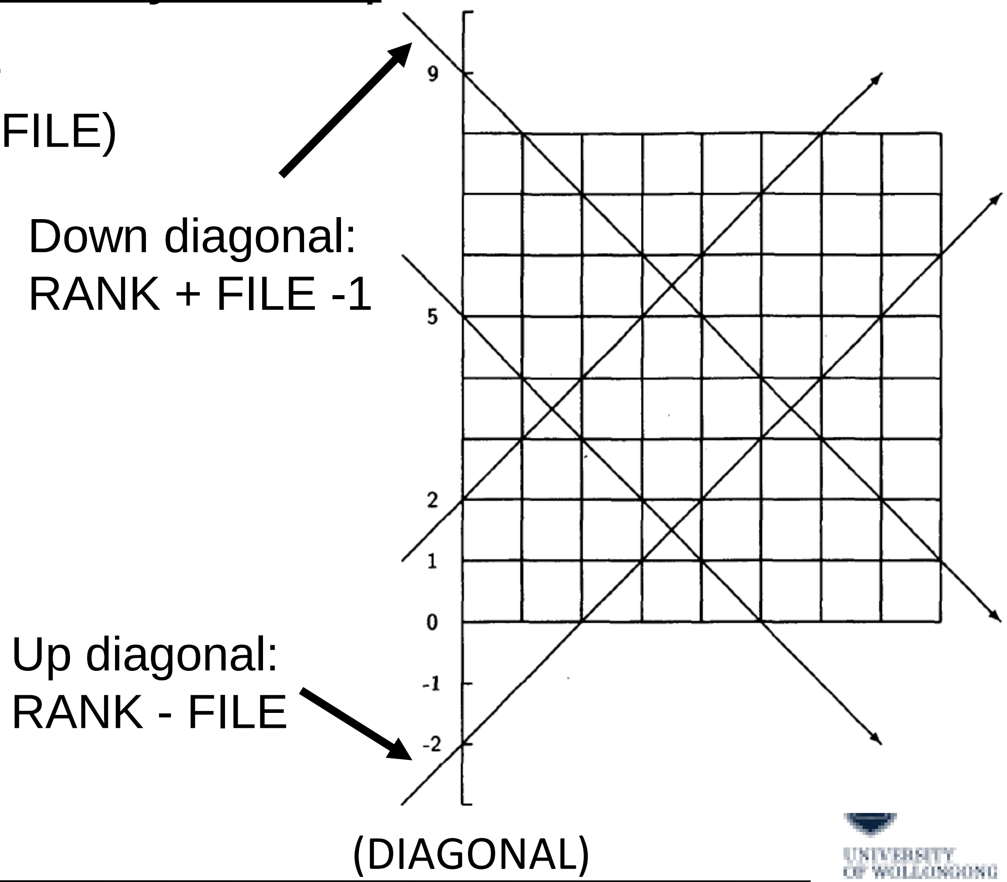


(DIAGONAL)

Eight queens

Problem: Eight queens must be placed on a chessboard so that no queen attacks any others. A chessboard is a square grid with eight columns, or files, and eight rows, or ranks. *<u>When a queen is placed on a square, it attacks any other queen that sits on the same rank, file, or diagonals.</u>*





Eight queens

$SIZE == 8$

$FILE == 1 \mathinner{\ldotp\ldotp} SIZE$

$RANK == 1 \mathinner{\ldotp\ldotp} SIZE$

$SQUARE == FILE \times RANK$

$DIAGONAL == 1 - SIZE \mathinner{\ldotp\ldotp} 2 * SIZE - 1$

$$
\begin{array}{|l}
up, down : SQUARE \rightarrow DIAGONAL \\
\hline
\forall f : FILE;\ r : RANK \bullet \\
\qquad up\,(f, r) = r - f \land \\
\qquad down\,(f, r) = r + f - 1
\end{array}
$$

Queen cannot be on the same RANK and the same FILE

Queen cannot be on the same DIAGONAL (both up and down)

Queens

$$
\begin{array}{|l}
squares : FILE \rightarrowtail\!\!\!\!\twoheadrightarrow RANK \\
\hline
\{\, squares \lhd up,\ squares \lhd down \,\} \subseteq SQUARE \rightarrowtail DIAGONAL
\end{array}
$$

A solution: $\{1 \mapsto 8, 2 \mapsto 4, 3 \mapsto 1, 4 \mapsto 3, 5 \mapsto 6, 6 \mapsto 2, 7 \mapsto 7, 8 \mapsto 5\}$

Not a solution (same rank) : $\{1 \mapsto 1, 2 \mapsto 4, 3 \mapsto 1, 4 \mapsto 3, 5 \mapsto 6, 6 \mapsto 2, 7 \mapsto 7, 8 \mapsto 5\}$

Not a solution (same diagonal) $\{1 \mapsto 1, 2 \mapsto 2, 3 \mapsto 3, 4 \mapsto 4, 5 \mapsto 5, 6 \mapsto 6, 7 \mapsto 7, 8 \mapsto 8\}$

Case Study: An Automated Billing System

Problem description

Software consulting firms generally deal with several clients where each client contracts out of project to the firm and receives a set of services related to the project.

An employee in the firm may work on multiple projects at any one time, with an interleaved work schedule. *A customer is billed at an hourly rate and an employee is paid at another hourly rate*. The focus of the problem is to develop **an automated billing system** that can be used both for *billing the customers for their projects and for calculating the salaries of employees*.

Additional requirements

1. A project employs one or more employees.

2. The hourly rate charged for projects is the same for all the projects and is assigned at the initiation of the project.

3. The hourly salary is the same for all employees in the firm. The salary is independent of the project(s) assigned to the employee.

Additional requirements

4. The estimated number of hours for completing a project is fixed for billing purposes, whether or not the project is completed by the deadline. If a project is not completed within its estimated time, *the customer who initiated this project will not be billed for the extra hours* required by the firm to complete the project. However, *employees who work on this project during the extra hours will be paid* according to their salary rate.

5. Depending on the rate of progress and the nature of a project, *employees may be assigned to project or be removed from a project*.

6. A project, once initiated, will not be terminated until it is completed.

Additional requirements

7. It must be possible to perform the following operations:

(1) Add a new employee to the firm.

(2) Add a new customer.

(3) Initiate a project (by a known customer).

(4) Assign an employee to a project.

(5) Release an employee from a project.

(6) Report the work done by an employee.

(7) Calculate the salary of an employee for a given month and year.

(8) Bill a customer for a given month and year.

The model

The requirements reveal that *Employee, Customer,* and *Project* are three **composite data types** to be modeled with a number of static and dynamic relationship among them.

Schema type or *Cartesian product type* may be used to construct the data model for *Employee, Project,* and *Customer.*

Z specification

Basic types

The identifiers for *employees, customers*, and *projects* are represented by three distinct basic types.

[*EMPLOYEE_ID, CUSTOMER_ID, PROJECT_ID*]

Date

Date is a triple tuple *(day, month, year)*

How to model DATE with Z specification?

Date

Date is a triple tuple *(day, month, year)*

Day == 1 .. 31

Month ::= *January* | *February* | *March* | *April* | *May* |
June | *July* | *August* | *September* | *October* |
November | *December*

Year==1949 .. 2999

Date ==*Day* X *Month* X *Year*

UNIVERSITY OF WOLLONGONG AUSTRALIA

Projection functions

We next specify three functions, i.e., *day, month,* and *year* to select the fields of date.

$$
\begin{array}{|l}
\text{day: Date} \rightarrow \text{Day} \\
\text{month: Date} \rightarrow \text{Month} \\
\text{year: Date} \rightarrow \text{Year} \\
\hline
\forall \text{date: Date} \bullet \exists \text{d:Day, m:Month, y:Year} \mid \\
\text{(d, m, y)} = \text{date} \bullet \text{day(date)} = \text{d} \wedge \\
\text{month(date)} = \text{m} \wedge \text{year(date)} = \text{y}
\end{array}
$$

Global constraint to validate the Date

$\forall$date: Date •

(month(date) $\in$ {April, June, September, November} $\Rightarrow$ day(date) <=30) $\wedge$

(month(date) = February $\Rightarrow$ day(date) <= 28) $\wedge$

((year(date) mod 4 = 0 $\wedge$ year(date) mod 100 $\neq$ 0) $\vee$ (year(date) mod 400 = 0) $\Rightarrow$

day(date) <= 29)

Work hours

An employee may work for a maximum of 24 hours
Per day. We therefore define *Hours* as an *enumerated type*.

$Hours$ == 0 .. 24

Time sheet

Combining *Date* and *Hours,* we define the data type *Timesheet,* which shows the dates and the number of hours worked by an employee during each day on a particular project.

$$TimeSheet == Date \rightarrowtail\!\!\!\!\!\!\!+\ Hours$$

Sum Timesheet

The function *sum_timesheet* computes sum for a given time sheet.

$$sum\_timesheet : TimeSheet \rightarrow \mathbb{N}$$

$$\forall tsh : TimeSheet \bullet$$
$$tsh = \emptyset \Rightarrow sum\_timesheet\ (tsh) = 0 \land$$
$$tsh \neq \emptyset \Rightarrow (\exists date : Date \mid date \in \mathrm{dom}\ tsh \bullet$$
$$sum\_timesheet\ (tsh) = tsh\ (date) +$$
$$sum\_timesheet\ (\{date\} \ndres tsh)$$

Work sheet

There is at most one (logical) time sheet for a project.

A work sheet records the time sheet for one project.

$$WorkSheet == PROJECT\_ID \pfun TimeSheet$$

Project hours

Total hours per particular project

$$project\_hours : WorkSheet \rightarrow\!\!\!\!\!+ \boxed{(PROJECT\_ID \rightarrow\!\!\!\!\!+ \mathbb{N})}$$

$$\forall work : WorkSheet \bullet project\_hours\ (work) =$$
$$\{\exists\ pid : PROJECT\_ID\ ;\ hrs : Hours\ |$$
$$pid \in \mathrm{dom}\ work \wedge$$
$$hrs = sum\_timesheet\ (work\ pid) \bullet (\ pid \mapsto hrs)\}$$

So, project_hours == Worksheet ; sum_timesheet

Sum work hours

$sum\_workhours : WorkSheet \mapsto \mathbf{N}$

$\forall work : WorkSheet \bullet$

$work = \varnothing \Rightarrow sum\_workhours(work) = 0 \land$

$work \neq \varnothing \Rightarrow (\exists pid : PROJECT\_ID \mid pid \in \mathrm{dom}\ work \bullet$

$sum\_worksheet\ (work) = sum\_timesheet\ (pid) +$

$sum\_\ workhours\ (\{pid\} \lhd work)$

State of the system

project_rate : $\mathbb{N}_1$

employee_rate : $\mathbb{N}_1$

bill_charge : $\mathbb{N}_1$

State space schema Organization

Organization

$employees : EMPLOYEE\_ID \mapsto WorkSheet$

$customers : CUSTOMER\_ID \mapsto WorkSheet$

$projects : PROJECT\_ID \to (\mathbf{N} \times \mathbf{N})$

expected
working hours

actual working
hours

Organization (continue)

$(\forall eid : EMPLOYEE\_ID \mid eid \in \mathrm{dom}\ employees \bullet$

$\mathrm{dom}(employees\ eid) \subseteq \mathrm{dom}\ projects \wedge$

$(\forall\ pid : PROJECT\_ID \mid pid \in \mathrm{dom}\ (\ employees\ eid\ ) \bullet$

$second\ (projects\ pid) \geq sum\_timesheet((employees\ eid\ )\ pid))) \wedge$

$(\forall cid : CUSTOMER\_ID \mid cid \in \mathrm{dom}\ customers \bullet$

$\mathrm{dom}\ (customers\ cid) \subseteq \mathrm{dom}\ projects \wedge$

$(\forall\ pid : PROJECT\_ID \mid pid \in \mathrm{dom}\ (customers\ cid) \bullet$

$second\ (projects\ pid) = sum\_timesheet((customers\ cid\ )pid))) \wedge$

$(\forall pid : PROJECT\_ID \mid pid \in \mathrm{dom}\ projects \bullet$

$first\ (projects\ pid) \leq second(\ projects\ pid))$

Initialization

InitOrganization

Organization'

$employees' = \Phi$

$customers' = \Phi$

$projects' = \Phi$

Operations

The operation *AddEmployee* adds a new employee to the organization, who is not yet assigned to any project.

AddEmployee

$\Delta organization$

$(\exists\ eid : EMPLOYEE\_ID \mid eid \in \mathrm{dom}\ employees \bullet$

$\quad employees' = employees \oplus \{ eid \mapsto \varnothing \})$

$customers' = customers$

$projects' = projects$

Employee not work on any particular project

Initiate Project

$$
\begin{array}{l}
\textit{InitiateProject} \\
\Delta \textit{Organization} \\
\textit{cid}? : \textit{CUSTOMER\_ID} \\
\textit{estimate}? : \mathbf{N} \\
\hline
\textit{cid}? \in \mathrm{dom}\ \textit{customers} \\
(\exists\ \textit{pid} : \textit{PROJECT\_ID} \mid \textit{pid} \notin \mathrm{dom}\ \textit{projects} \bullet \\
\quad \textit{customers}' = \textit{customers} \oplus \{\ \textit{cid}? \mapsto \\
\qquad \underline{(\textit{customers cid}?) \oplus \{\ \textit{pid} \mapsto \varnothing\ \}}\} \wedge \\
\quad \textit{projects}' = \textit{projects} \oplus \{\ \textit{pid} \mapsto (\textit{estimate}?\ ,0)\}) \\
\textit{employees}' = \textit{employees}
\end{array}
$$

Update customer set who owns existing project

Assign Employee

AssignEmployee

$\Delta Organization$

$eid? : EMPLOYEE\_ID$

$pid? : PROJECT\_ID$

$eid? \in \mathrm{dom}\ employees$

$pid? \in \mathrm{dom}\ projects$

$\neg\ (pid? \in \mathrm{dom}\ (\ employees\ eid?\ ))$

$employees' = employees \oplus \{eid? \mapsto (employees\ eid?) \oplus \{pid? \mapsto \varnothing\}\}$

$customers' = customers$

$projects' = projects$

Update employee set who is assigned to a project

Calculate Salary

CalculateSalary

Ξ*Organization*
eid?: *EMPLOYEE_ID*
month?: *Month*
year?: *Year*
salary!: **N**

---

eid? $\in$ **dom** *employees*
(**let** *worksheet*==*select_timesheet* ((*employees eid*?), *month*?, *year*?) • *salary*!= *sum_workhours*(*worksheet*) * *employee_rate*)