

# Z SPECIFICATIONS

Formal Methods

BCS 2213

Semester 2 Session 2014/2015

# Schema Calculus

Allows structure specifications by building large schemas from smaller ones. Leads to separation of concerns:

- ▣ separating normal operations from error handling
- ▣ separating access restrictions from functional behaviors
- ▣ etc.

# Schema calculus

3

Provide operations for combining schemas, e.g.,

$S_1 \wedge S_2$   
where  $S_1$  and  $S_2$  are schemas

## Two main operations:

- $\Lambda$  - logical conjunction of the two predicate parts. Any common variables of the two schemas are merged
- $\vee$  - the result is a schema in which the predicate part is disjunction of the predicate parts of its two arguments.

# Schema Calculus

- Merge declarations  $D$  and combine predicates  $P$

$$S_1 \triangleq [D_1 \mid P_1]$$

$$S_2 \triangleq [D_2 \mid P_2]$$

$$S_1 \wedge S_2 \equiv [D_1; D_2 \mid P_1 \wedge P_2]$$

# Example

Quotient

$n, d, q, r: \mathbb{N}$

$d \neq 0$

$n = q * d + r$

Remainder

$r, d: \mathbb{N}$

$r < d$

Division  $\triangleq$  Quotient  $\wedge$  Remainder

Division

$n, d, q, r: \mathbb{N}$

$d \neq 0$

$r < d$

$n = q * d + r$

# Race condition

6

We have not handled the condition when user tries to add a birthday, which is already known to the system, or tries to find the birthday of someone not known.

Lets handle this by adding an extra **result!** parameter to each operation.

result! will have the type (*freetype*) **REPORT**

**REPORT** ::= ok | already\_known | not\_known

# Further development of BirthdayBook

*Freetype definition ::=*

*REPORT ::= ok | already\_known | not\_known*

*Success*

*result! : REPORT*

*result! = ok*

*AddBirthday*

*$\Delta$ BirthdayBook*

*name? : NAME*

*date? : DATE*

*name?  $\notin$  known*

*birthday' = birthday  $\cup$  {name?  $\mapsto$  date?}*

# Using Logical Conjunction Operator

8

Lets combine by conjunction operator  $\wedge$  two schemas

**AddBirthday  $\wedge$  Success**

The result is an operation which acts as *AddBirthday* and produces the *output* **ok** (for correct input).



# Strengthening *AddBirthday*

*AlreadyKnown*

$\exists$  *BirthdayBook*

*name?* : *NAME*

*result!* : *REPORT*

*name?*  $\in$  *known*

*result!* = *already\_known*

This declaration specifies that if error occurs, the state of the system should not change.

**Robust** version of *AddBirthday* can be

$RAddBirthday \triangleq$

$(AddBirthday \wedge Success) \vee AlreadyKnown$

Note, in CZT for the definition symbol you can use ==

# *RAddBirthday*

*Success*

*result!* : *REPORT*

*result!* = *ok*

*AddBirthday*

$\Delta$ *BirthdayBook*

*name?* : *NAME*

*date?* : *DATE*

*name?*  $\notin$  *known*

*birthday'* = *birthday*  $\cup$  {*name?*  $\mapsto$  *date?*}

*AlreadyKnown*

$\exists$  *BirthdayBook*

*name?* : *NAME*

*result!* : *REPORT*

*name?*  $\in$  *known*

*result!* = *already\_known*

*RAddBirthday*  $\hat{=}$

(*AddBirthday*  $\wedge$  *Success*)  $\vee$  *AlreadyKnown*

*RAddBirthday*

$\Delta$ *BirthdayBook*

*name?* : *NAME*

*date?* : *DATE*

*result!* : *REPORT*

(*name?*  $\notin$  *known*  $\wedge$   
*birthday'* = *birthday*  $\cup$  {*name?*  $\mapsto$  *date?*}  $\wedge$   
*result!* = *ok*)  $\vee$

(*name?*  $\in$  *known*  $\wedge$   
*birthday'* = *birthday*  $\wedge$   
*result!* = *already\_known*)

Notice the  
framing  
constraint. Why?

# Strengthening *FindBirthday* and *Remind*

*FindBirthday*

$\exists \text{BirthdayBook}$

$\text{name?} : \text{NAME}$

$\text{date!} : \text{DATE}$

$\text{name?} \in \text{known}$

$\text{date!} = \text{birthday}(\text{name?})$

*Remind*

$\exists \text{BirthdayBook}$

$\text{today?} : \text{DATE}$

$\text{cards!} : \mathbb{P} \text{ NAME}$

$\text{cards!} = \{ n : \text{known} \mid \text{birthday}(n) = \text{today?} \}$

# *RFindBirthday* and *RRemind*

*REPORT* ::= *ok* | *already\_known* | *not\_known*

*NotKnown*

$\exists \text{BirthdayBook}$

*name?* : *NAME*

*result!* : *REPORT*

*name?*  $\notin$  *known*

*result!* = *not\_known*

*RFindBirthday*  $\cong$  (*FindBirthday*  $\wedge$  *Success*)  $\vee$  *NotKnown*

*RRemind*  $\cong$  *Remind*  $\wedge$  *Success*

# From Specification to Designs and Implementation

- Previously, we use Z to specify a software module
- Now, lets use Z to design a program
- Key idea: *data refinement*
  - ▣ Describe concrete data structures (vs abstract data in specification)
  - ▣ Derive descriptions of operations in terms of concrete data structures

Data refinement leads to operation refinement and algorithm development

# From specification to design

14

## Data Refinement

“ to describe the concrete data structures which the program will use to represent the abstract data in the specification, and to derive description of the operations in terms of the concrete data structures”

Direct Refinement: method to go directly from abstract specification to program in one step

# Implementation of Birthday Book

- Representation in concrete data structure. A possible representation in c language

*NAME* names[10];

*DATE* dates[10];

*BirthdayBook*

*known* :  $\mathbb{P}$  *NAME*

*birthday* : *NAME*  $\rightarrow$  *DATE*

*known* = dom *birthday*

# Concrete State Model - *BirthdayBook1*

- Arrays mathematically are functions:

$names : \mathbb{N}_1 \rightarrow NAME$

$dates : \mathbb{N}_1 \rightarrow DATE$

- $names[i]$  as  $names(i)$

- $names[i] = v$  as  $names' = names \oplus \{i \mapsto v\}$

$\oplus$  overriding operation

*BirthdayBook1*

$names : \mathbb{N}_1 \rightarrow NAME$

$dates : \mathbb{N}_1 \rightarrow DATE$

$hwm : \mathbb{N}$

$\forall i, j : 1..hwm \bullet$

$i \neq j \Rightarrow names(i) \neq names(j)$



# Override operation

17

**$\text{names}' = \text{names} \oplus \{i \longrightarrow v\};$**

It is the same as

**$\text{names}[i] := v;$**

the right side of this equation is a function which takes the same value as **names** everywhere except at the argument **i**, where it takes the value **v**.

# Operation Refinement, *AddBirthday1*

- Manipulate *names* and *dates* arrays

*AddBirthday1*

$\Delta \text{BirthdayBook1}$

$\text{name?} : \text{NAME}$

$\text{date?} : \text{DATE}$

$\forall i : 1..hwm \bullet \text{name?} \neq \text{names}(i)$

$hwm' = hwm + 1$

$\text{names}' = \text{names} \oplus \{hwm' \mapsto \text{name?}\}$

$\text{dates}' = \text{dates} \oplus \{hwm' \mapsto \text{date?}\}$

*BirthdayBook1*

$\text{names} : \mathbb{N}_1 \rightarrow \text{NAME}$

$\text{dates} : \mathbb{N}_1 \rightarrow \text{DATE}$

$hwm : \mathbb{N}$

$\forall i, j : 1..hwm \bullet$

$i \neq j \Rightarrow \text{names}(i) \neq \text{names}(j)$

# Implementation of *AddBirthday1*

```
void addBirthday(NAME name, DATE date) {  
    hwm++;  
    names[hwm] = name;  
    dates[hwm] = date;  
}
```

*AddBirthday1* \_\_\_\_\_

$\Delta BirthdayBook1$

$name? : NAME$

$date? : DATE$

$\forall i : 1..hwm \bullet name? \neq names(i)$

$hwm' = hwm + 1$

$names' = names \oplus \{hwm' \mapsto name?\}$

$dates' = dates \oplus \{hwm' \mapsto date?\}$

# Refinement of *FindBirthday*

*FindBirthday1*

$\exists \textit{BirthdayBook1}$

$\textit{name?} : \textit{NAME}$

$\textit{date!} : \textit{DATE}$

$\exists i : 1..hwm \bullet$

$\textit{name?} = \textit{names}(i) \wedge \textit{date!} = \textit{dates}(i)$

*BirthdayBook1*

$\textit{names} : \mathbb{N}_1 \rightarrow \textit{NAME}$

$\textit{dates} : \mathbb{N}_1 \rightarrow \textit{DATE}$

$\textit{hwm} : \mathbb{N}$

$\forall i, j : 1..hwm \bullet$

$i \neq j \Rightarrow \textit{names}(i) \neq \textit{names}(j)$

# Refinement of *Remind*

*Remind1*

$\exists \text{BirthdayBook1}$

$\text{today?} : \text{DATE}$

$\text{cardlist!} : \mathbb{N}_1 \rightarrow \text{NAME}$

$\text{ncards!} : \mathbb{N}$

$\{ i : 1..ncards! \bullet \text{cardlist!}(i) \}$   
 $= \{ j : 1..hwm \mid \text{dates}(j) = \text{today?} \bullet \text{names}(j) \}$

*AbsCards*

$\text{cards} : \mathbb{P} \text{NAME}$

$\text{cardlist} : \mathbb{N}_1 \rightarrow \text{NAME}$

$\text{ncards} : \mathbb{N}$

$\text{cards} = \{ i : 1..ncards \bullet \text{cardlist}(i) \}$

# Refinement of *InitBirthdayBook*

<i>InitBirthdayBook1</i>	_____
<i>BirthdayBook1</i>	
<i>hwm</i> = 0	

# Abstraction Relation

- Relation between abstract state space and concrete state space, BirthdayBook and BirthdayBook1

*Abs*

*BirthdayBook*

*BirthdayBook1*

$known = \{ i : 1..hwm \bullet names(i) \}$

$\forall i : 1..hwm \bullet$

$birthday(names(i)) = dates(i)$

*BirthdayBook*

$known : \mathbb{P} NAME$

$birthday : NAME \rightarrow DATE$

$known = \text{dom } birthday$

*BirthdayBook1*

$names : \mathbb{N}_1 \rightarrow NAME$

$dates : \mathbb{N}_1 \rightarrow DATE$

$hwm : \mathbb{N}$

$\forall i, j : 1..hwm \bullet$

$i \neq j \Rightarrow names(i) \neq names(j)$

# Example(Data and Direct Refinement)

24

FindBirthday1

$\equiv$  BirthdayBook1

name?:NAME

date?:DATE

$\exists i : 1.. hwm$

name?=names(i)  $\wedge$  date! = dates(i)

Procedure FindBirthday(name: NAME; var date : DATE);

var i: INTEGER;

begin

i:=1;

while names[i]  $\neq$  name do i := i+1;

dates := dates[i]

end;



# Features Notation

25

- Is used to verify the specification
- Independent of program code
- Use mathematical model of data
- Allow to model a specification which can directly lead to the code.
- Represent both static and dynamic aspects of a system

# Features Notation

26

- ❑ Decompose specification into small pieces (Schemas)
- ❑ Schemas are used to describe both static and dynamic aspects of a system
- ❑ Data Refinement
- ❑ Direct Refinement
- ❑ You can ignore details in order to focus on the aspects of the problem you are interested in
- ❑ ISO standard, ISO/IEC 13568:2002