Z SPECIFICATIONS

Formal Methods

BCS 2213

Semester 1 Session 2014/2015

Schema Calculus

Allow structure specifications by building large schemas from smaller ones:

- Separating normal operations from error handling
- Separating access restrictions from functional behaviors
- Framing operations

Lead to separation of concerns

Schema calculus

Provide operations for combining schemas, e.g.,

$$S_1 \wedge S_2$$

where S_1 and S_2 are schemas

Two main operations:

Λ - logical conjunction of the two predicate parts. Any common variables of the two schemas are merged

V - 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

$S_1 \triangleq [D_1 \mid P_1]$

$S_2 \triangleq [D_2 \mid P_2]$

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

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

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.

Handle this by adding an extra result! to each operation.

Result := ok| already_known | not_known

Further development of BirthdayBook

Freetype definition **::=**

Result := ok| already_known | not_known

Success

$result! : REPORT$

$result! = ok$

AddBirthday

$\Delta BirthdayBook$
$name? : NAME$
$date? : DATE$

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

Example of Logical Conjunction Operator

Lets combine by conjunction operator Λ two schemas

AddBirthday Λ Success

The result is an operation which, for correct *input*, both acts as described by *AddBirthday* and produces the *output* ***ok****.*

Strengthening *AddBirthday*

$$
\begin{array}{|l}
\hline
\textit{AlreadyKnown} \\
\Xi \textit{BirthdayBook} \\
\textit{name}? : \textit{NAME} \\
\textit{result}! : \textit{REPORT} \\
\hline
\textit{name}? \in \textit{known} \\
\textit{result}! = \textit{already\_known} \\
\hline
\end{array}
$$

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

Robust version of AddBirthday can be

$$
\begin{array}{l}
\textit{RAddBirthday} \mathrel{\widehat{=}} \\
\quad (\textit{AddBirthday} \land \textit{Success}) \lor \textit{AlreadyKnown}
\end{array}
$$

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

RAddBirthday

Success

$$
\begin{array}{l}
result! : REPORT \\
\hline
result! = ok
\end{array}
$$

AddBirthday

$$
\begin{array}{l}
\Delta BirthdayBook \\
name? : NAME \\
date? : DATE \\
\hline
name? \notin known \\
birthday' = birthday \cup \{name? \mapsto date?\}
\end{array}
$$

AlreadyKnown

$$
\begin{array}{l}
\Xi BirthdayBook \\
name? : NAME \\
result! : REPORT \\
\hline
name? \in known \\
result! = already\_known
\end{array}
$$

$$
RAddBirthday \mathrel{\widehat{=}} (AddBirthday \land Success) \lor AlreadyKnown
$$

RAddBirthday

$$
\begin{array}{l}
\Delta BirthdayBook \\
name? : NAME \\
date? : DATE \\
result! : REPORT \\
\hline
(name? \notin known \land \\
\quad birthday' = birthday \cup \{name? \mapsto date?\} \land \\
\quad result! = ok) \lor \\
(name? \in known \land \\
\quad birthday' = birthday \land \\
\quad result! = already\_known)
\end{array}
$$

Notice the framing constraint. Why?

Strengthening *FindBirthday* and *Remind*

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{FindBirthday}} \\
\hline
\Xi \mathit{BirthdayBook} \\
\mathit{name}? : \mathit{NAME} \\
\mathit{date}! : \mathit{DATE} \\
\hline
\mathit{name}? \in \mathit{known} \\
\mathit{date}! = \mathit{birthday}(\mathit{name}?) \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{Remind}} \\
\hline
\Xi \mathit{BirthdayBook} \\
\mathit{today}? : \mathit{DATE} \\
\mathit{cards}! : \mathbb{P}\ \mathit{NAME} \\
\hline
\mathit{cards}! = \{\, n : \mathit{known} \mid \mathit{birthday}(n) = \mathit{today}? \,\} \\
\hline
\end{array}
$$

RFindBirthday and *RRemind*

REPORT ::= *ok* | *already_known* | *not_known*

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{NotKnown}} \\
\Xi \textit{BirthdayBook} \\
\textit{name}? : \textit{NAME} \\
\textit{result}! : \textit{REPORT} \\
\hline
\textit{name}? \notin \textit{known} \\
\textit{result}! = \textit{not\_known} \\
\hline
\end{array}
$$

$$\textit{RFindBirthday} \mathrel{\widehat=} (\textit{FindBirthday} \land \textit{Success}) \lor \textit{NotKnown}$$

$$\textit{RRemind} \mathrel{\widehat=} \textit{Remind} \land \textit{Success}$$

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 or algorithm development

From specification to design

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 operation 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. One possible representation in c

  NAME names[10];

  DATE dates[10];

BirthdayBook

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

$$birthday : NAME \rightarrow DATE$$

$$known = \operatorname{dom} birthday$$

Concrete State Model - *BirthdayBook1*

- Arrays modeled mathematically as 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

$$\begin{array}{|l}
\textit{BirthdayBook1} \\
\hline
names : \mathbb{N}_1 \rightarrow NAME \\
dates : \mathbb{N}_1 \rightarrow DATE \\
hwm : \mathbb{N} \\
\hline
\forall\, i, j : 1 \,..\, hwm \bullet \\
\quad i \neq j \Rightarrow names(i) \neq names(j) \\
\hline
\end{array}$$

Override operation

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

$names[i] := v;$

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'.

Abstraction Relation

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

Abs

$$BirthdayBook$$
$$BirthdayBook1$$

$$known = \{\, i : 1 \,..\, hwm \bullet names(i) \,\}$$
$$\forall\, i : 1 \,..\, hwm \bullet$$
$$\quad birthday(names(i)) = dates(i)$$

BirthdayBook

$$known : \mathbb{P}\ NAME$$
$$birthday : NAME \rightarrow DATE$$

$$known = \operatorname{dom} birthday$$

BirthdayBook1

$$names : \mathbb{N}_1 \rightarrow NAME$$
$$dates : \mathbb{N}_1 \rightarrow DATE$$
$$hwm : \mathbb{N}$$

$$\forall\, i, j : 1 \,..\, hwm \bullet$$
$$\quad i \neq j \Rightarrow names(i) \neq names(j)$$

Operation Refinement, *AddBirthday1*

- Manipulate *names* and *dates* arrays

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{AddBirthday1}} \\
\hline
\Delta \textit{BirthdayBook1} \\
\textit{name}? : \textit{NAME} \\
\textit{date}? : \textit{DATE} \\
\hline
\forall\, i : 1 \mathinner{\ldotp\ldotp} \textit{hwm} \bullet \textit{name}? \neq \textit{names}(i) \\
\textit{hwm}' = \textit{hwm} + 1 \\
\textit{names}' = \textit{names} \oplus \{\textit{hwm}' \mapsto \textit{name}?\} \\
\textit{dates}' = \textit{dates} \oplus \{\textit{hwm}' \mapsto \textit{date}?\} \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{BirthdayBook1}} \\
\hline
\textit{names} : \mathbb{N}_1 \to \textit{NAME} \\
\textit{dates} : \mathbb{N}_1 \to \textit{DATE} \\
\textit{hwm} : \mathbb{N} \\
\hline
\forall\, i, j : 1 \mathinner{\ldotp\ldotp} \textit{hwm} \bullet \\
\quad i \neq j \Rightarrow \textit{names}(i) \neq \textit{names}(j) \\
\hline
\end{array}
$$

Implementation of *AddBirthday1*

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

AddBirthday1

$\Delta BirthdayBook1$
$name? : NAME$
$date? : DATE$

$\forall\, i : 1 \mathinner{..} hwm \bullet name? \neq names(i)$
$hwm' = hwm + 1$
$names' = names \oplus \{hwm' \mapsto name?\}$
$dates' = dates \oplus \{hwm' \mapsto date?\}$

Refinement of *FindBirthday*

$$
\begin{array}{|l}
\hline
\textit{FindBirthday1} \\
\Xi \textit{BirthdayBook1} \\
\textit{name?} : \textit{NAME} \\
\textit{date!} : \textit{DATE} \\
\hline
\exists\, i : 1 \mathinner{\ldotp\ldotp} \textit{hwm} \bullet \\
\quad \textit{name?} = \textit{names}(i) \;\land\; \textit{date!} = \textit{dates}(i) \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
\textit{BirthdayBook1} \\
\textit{names} : \mathbb{N}_1 \to \textit{NAME} \\
\textit{dates} : \mathbb{N}_1 \to \textit{DATE} \\
\textit{hwm} : \mathbb{N} \\
\hline
\forall\, i, j : 1 \mathinner{\ldotp\ldotp} \textit{hwm} \bullet \\
\quad i \neq j \Rightarrow \textit{names}(i) \neq \textit{names}(j) \\
\hline
\end{array}
$$

Refinement of *Remind*

$$
\begin{array}{l}
\textit{Remind1} \\
\hline
\Xi \textit{BirthdayBook1} \\
\textit{today?} : \textit{DATE} \\
\textit{cardlist!} : \mathbb{N}_1 \rightarrow \textit{NAME} \\
\textit{ncards!} : \mathbb{N} \\
\hline
\{\, i : 1 \mathinner{\ldotp\ldotp} \textit{ncards!} \bullet \textit{cardlist!}(i) \,\} \\
\quad = \{\, j : 1 \mathinner{\ldotp\ldotp} \textit{hwm} \mid \textit{dates}(j) = \textit{today?} \bullet \textit{names}(j) \,\} \\
\hline
\end{array}
$$

$$
\begin{array}{l}
\textit{AbsCards} \\
\hline
\textit{cards} : \mathbb{P}\ \textit{NAME} \\
\textit{cardlist} : \mathbb{N}_1 \rightarrow \textit{NAME} \\
\textit{ncards} : \mathbb{N} \\
\hline
\textit{cards} = \{\, i : 1 \mathinner{\ldotp\ldotp} \textit{ncards} \bullet \textit{cardlist}(i) \,\} \\
\hline
\end{array}
$$

Refinement of *InitBirthdayBook*

$InitBirthdayBook1$
$BirthdayBook1$
$hwm = 0$

Example(Data and Direct Refinement)

FindBirthday1

Ξ BirthdayBook1

name?:NAME

date?:DATE

$\exists i : 1..\ hwm$

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

```
Procedure FindBirthday(name: NAME; var date : DATE);
            var i: INTEGER;
      begin
            i:=1;
            while names[i] ≠ name do i := i+1;
            dates := dates[i]
      end;
```

Features Z Notation

- 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 Z Notation

- 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