FMSE: Lecture 3

The Specification Language Z:

Preconditions, Robustness, Composition

Goals of Lecture 3

At the end of this lecture you should be able to:

- give the precondition of an operation
- understand the role of preconditions in the analysis of specifications
- make operations robust
- use schema composition in structuring a specification

The Library System

$$maxloan : \mathbb{N}_1$$

$$[TITLE, COPY, READER]$$

$$title : COPY \longrightarrow TITLE$$

Library

$$collection : \mathbb{P}\, COPY$$
$$readers : \mathbb{P}\, READER$$
$$issued : COPY \nrightarrow READER$$

$$\mathrm{dom}\, issued \subseteq collection$$
$$\mathrm{ran}\, issued \subseteq readers$$
$$\forall\, r : \mathrm{ran}\, issued \bullet \#\{c : COPY \mid issued(c) = r\} \leq maxloan$$

Issueing a Copy

Issue

$\Delta Library$

$r? : READER$

$c? : COPY$

$issued' = issued \cup \{(c?, r?)\}$

$collection' = collection$

$readers' = readers$

Note that the invariants should hold after this operation.

When can this operation be applied?

Since we know that the invariants should be true after the operation:

- $\mathrm{dom}\, issued' \subseteq collection'$ so $c? \in collection$
- $\mathrm{ran}\, issued' \subseteq readers'$ so $r? \in readers$
- $\#\{c : COPY \mid issued'(c) = r?\} \leq maxloan$ so $\#\{c : COPY \mid issued(c) = r?\} \leq maxloan$
- $issued'$ is a function, so $c? \notin \mathrm{dom}\, issued$

It is good practice to explicitly state these preconditions in the operation schema! So:

Issueing with explicit preconditions

Issue

$\Delta Library$

$r? : READER$

$c? : COPY$

$r? \in readers$

$c? \in collection \setminus (\mathrm{dom}\, issued)$

$\#\{c : COPY \mid issued(c) = r?\} < maxloan$

$issued' = issued \cup \{(c?, r?)\}$

$collection' = collection$

$readers' = readers$

Precondition: Formal Definition

It may be the case that not all preconditions are explicitly present in a schema. Generally they must be *calculated*.

The precondition is the condition that has to be fulfilled in order for the state after the operation to exist. So the precondition is equivalent to saying: there is a valid state after the operation.

So the precondition of an operation schema *Operation* over a system with state schema *State* is expressed by

$$\exists\, State' \bullet Operation$$

Calculating the Precondition

The precondition predicate can be evaluated by expansion and simplification by logical calculation.

- this evaluation is often difficult and asks for experience in logic
- but it can be tool supported (see Z/EVES)
- we often calculate the preconditions by informal reasoning (like in the library example)

Consistency of Initialisation

Similar to the precondition of an operation we may want to prove that for an initialisation *Init* of a system *State*, there is an initial state that satisfies *Init*.

This is expressed by

$$\exists\, State \bullet Init$$

Note: some people look at *Init* as an operation with only a final state and no begin state. Therefore they import $State'$ in *Init*, instead of *State*.

Design Aspects of Preconditions

- explicit preconditions clarify operations
- preconditions can be found by tool supported analysis

If a state does not satisfy the precondition the effect of the operation is not definied....

- a *robust* operation is always applicable (so has precondition *true*).
- precondition analysis can check robustness
- robust operations often incorporate error conditions.

Error Conditions

The precondition of *Issue* is not fulfilled when one of the following error conditions holds:

- $c? \notin \mathit{collection} \setminus \operatorname{dom} \mathit{issued}$
- $r? \notin \mathit{readers}$
- $\#\{c : \mathit{COPY} \mid \mathit{issued}(c) = r?\} = \mathit{maxloan}$

We choose to leave the Library unchanged if any of these error conditions hold.

(but with some fantasy you can think of alternative actions for each error condition).

Monolithic Robust Operation

Issue

$\Delta Library$

$r? : READER$

$c? : COPY$

$(r? \in readers \wedge c? \in collection \setminus (\mathrm{dom}\, issued) \wedge$

$\#\{c : COPY \mid issued(c) = r?\} < maxloan \wedge$

$issued' = issued \cup \{(c?, r?)\} \wedge collection' = collection$

$\wedge\ readers' = readers) \vee ((r? \notin readers \vee$

$c? \notin collection \vee c? \in \mathrm{dom}\, issued \vee$

$\#\{c : COPY \mid issued(c) = r?\} = maxloan) \wedge issued' = issued$

$\wedge\ collection' = collection \wedge readers' = readers)$

Monolithic Operation (cont'd)

The monolithic operation is ugly and big and difficult to read!

Is there a way to do it in a nicer and more structured way?

Yes: using schema composition.

(we deal with schema disjunction and conjunction)

Schema Conjunction

Consider the schemas:

Quotient

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

$$d \neq 0$$
$$n = q * d + r$$

Remainder

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

$$r < d$$

Schema Conjunction (cont'd)

These may be combined using *schema conjunction* to form:

$$Division \mathrel{\widehat{=}} Quotient \land Remainder$$

What is the formal mening of *Division*?

Schema Conjunction: Semantics

The conjunction of two schemas is equivalent to a schema with:

- the *union* of the declarations of the two schemas, and
- the *conjunction* of the predicates of the two schemas.

An equivalent (expanded) definition of *Division* thus is:

Division

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

$d \neq 0$
$r < d$
$n = q * d + r$

Schema Disjunction

The *disjunction* of two schemas is equivalent to a schema with:

- the *union* of the declarations of the two schemas, and
- the *disjunction* of the predicates of the two schemas.

Example: Given the following schema for divison by zero:

DivideByZero

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

$$d = 0 \land q = 0 \land r = 0$$

the total operation for division is now given by:

$$T\_Division \mathrel{\widehat{=}} Division \lor DivideByZero$$

Schema Disjunction (cont'd)

An equivalent definition for $T\_Division$ in a single schema (the expanded version) is:

$T\_Division$

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

$(d \neq 0 \land r < d \land n = q * d + r) \lor$
$(d = 0 \land q = 0 \land r = 0)$

A Structured Issue Operation

$$
\begin{aligned}
IssueTotal \mathrel{\widehat=} {} & Issue \lor WrongReader \lor \\
& WrongCopy \lor LimitReached
\end{aligned}
$$

where

WrongReader

$\Xi Library$
$r? : READER$
$c? : COPY$

$r? \notin readers$

WrongCopy

$\Xi Library$

$r? : READER$

$c? : COPY$

$c? \notin collection \setminus \operatorname{dom} issued$

LimitReached

$\Xi Library$

$r? : READER$

$c? : COPY$

$\#\{c : COPY \mid issued(c) = r?\} = maxloan$

Adding Messages

It is a good idea to add messages that explicitly show in which situation we are.

For this *Issue* operation we define the following type:

$$\begin{aligned} MESSAGE ::= {} & OK \mid unregistered\_reader \mid \\ & copy\_unavailable \mid limit\_reached \end{aligned}$$

(this is a global type, placed at the top of the specification)

Adding Types (cont'd)

$$\begin{aligned}
IssueTotal \mathrel{\widehat{=}} {} & (Issue \land M_ok) \lor \\
& (WrongReader \land M_wr) \lor \\
& (WrongCopy \land M_wc) \lor \\
& (LimitReached \land M_lr)
\end{aligned}$$

where

M_ok

$m! : MESSAGE$

$m! = ok$

M_wr

$m! : MESSAGE$

$m! = unregisterd\_reader$

M_wc

$m! : MESSAGE$

$m! = copy\_unavailable$

M_lr

$m! : MESSAGE$

$m! = limit\_reacheD$

Conclusions

- better give explicit preconditions for operations
- the precondition can be calculated
- a robust operation is always applicable, i.e. it has precondition *true*
- operations can be made robust by incorporating error conditions
- this can be done in a structured way using schema composition
- these structuring facilities are especially important for large specifications