Chapter 12: Immanent Reasoning

Combining Z specifications and proofs

Class1

d:P Person

d <= Max

ΔClass1 $\triangleq$ Class1 $\wedge$ Class1 '

ΞClass1 $\triangleq$ [ΔClass1 | d' = d]

InitClass1 $\triangleq$ [Class1 | d = $\varnothing$]

Operations:

Enter1

ΔClass1
p?: Person

d < Max
p?$\notin$ d
d' = d $\cup${p?}

Operations:

Leave1

ΔClass1
p?: Person

p?$\in$ d
d' = d \ {p?}

Schema names as part of a formula

- $\exists$d: P Person | #d <= Max • d = $\varnothing$
  may be written as $\exists$Class1 • d = $\varnothing$
- $\forall$d: P Person | #d <= Max • d $\subseteq$ male
  may be written as $\forall$Class1 • d $\subseteq$ male
- λd: P Person | #d <= Max • #d
  may be written as λClass1 • #d
- $\exists$d': P Person | #d' <= Max • #d' <= Max $\wedge$ d' = $\varnothing$
  may be written as $\exists$d': P Person | #d' <= Max • InitClass1'

Initial State Proof Obligation:

- One of the things that we need to prove or at least convince ourselves of in some other way, about every specification that we write is that an initial state exists.
- In the case of the classroom system this proof obligation is expressed by the formula $\exists$Class1' • InitClass1'

$\exists$Class1' • InitClass1'

$\equiv$ <by definition>

$\exists$d': P Person | #d' <= Max • #d' <= Max $\wedge$ d' = $\varnothing$

$\equiv$ < restricted existential is equivalent to the unrestricted form>

$\exists$d': P Person • #d' <= Max $\wedge$ #d' <= Max $\wedge$ d' = $\varnothing$

$\exists d'$: P Person • **#d' <= Max ∧ #d' <= Max** ∧ d' = ∅
≡ < conjunction is idempotent >
$\exists d'$: P Person • **#d' <= Max** ∧ d' = ∅

≡ < ∃ Introduction >
#d' <= Max ∧ d' = ∅ **[∅ / d']** where the variable d' is of type P person

≡ < ∃ Introduction >
∅ <= Max ∧ ∅ = ∅
≡ True

Constructing Theories about Specifications

- Given a specification we can prove many things about it, constructing a theory of that specification, increasing our understanding and reinforcing our faith in its correctness.
- **The classroom specification:**
  - If someone leaves the classroom and re-enters it, the state of the classroom should be unaltered.
  - (Leave1; Enter1) ⇔ (Ξ Class1 | p? ∈ d)

Ξ Class1 | p? ∈ d
≡ <by definition>
(#d <= Max ∧ #d' <= Max ∧ d' = d ∧ p? ∈ d)

(Leave1; Enter1) ⇔ (Ξ Class1 | p? ∈ d)
≡ (Leave1; Enter1) ⇔ (#d <= Max ∧ #d' <= Max ∧ d' = d ∧ p? ∈ d)
Let Alpha ≙ Leave1; Enter1

```
── Alpha ──────────────
 d,d':P Person
 p?: Person
───────────
 # d <= Max
 # d' <= Max
 p?∈ d
 d' = d
───────────────────────
```

- This is identical to (Ξ Class1 | p? ∈ d) as required

Operations:

```
── Enter1 ─────────────
 ΔClass1
 p?: Person
───────────
 # d < Max
 p?∉ d
 d' = d ∪{p?}
───────────────────────
```


Operations:

```
── Leave1 ─────────────
 ΔClass1
 p?: Person
───────────
 p? ∈ d
 d' = d \ {p?}
───────────────────────
```


Investigating Preconditions

- Z contains a precondition operator pre which makes a schema out of a schema by hiding all the after and output variables of the given schema.
- Therefore, since Enter1 is a schema then so is PreEnter1
- Given a schema S, PreS indicates those states on which the operation S can be successfully carried out. So PreEnter1 tells us on which states Enter1 can be performed.
- Precondition schemas can usually be simplified and their existential quantifier removed.

Operations:

Enter1

ΔClass1
p?: Person

$\# d < Max$
$p? \notin d$
$d' = d \cup \{p?\}$

Investigating Preconditions

PreEnter1

d: P Person
p?: Person

$\exists d' : \mathbb{P}\ Person \bullet (\#d <= Max \wedge$
$\#d' <= Max$
$\#d < Max \wedge$
$p? \notin d \wedge$
$d' = d \cup \{p?\})$

- One Point Rule: Substitute $d \cup \{p?\}$ for d', removing $\exists$

Investigating Preconditions

PreEnter1

d: P Person
p?: Person

$\#d <= Max \wedge$
$\#(d \cup \{p?\}) <= Max \wedge$
$\#d < Max \wedge$
$p? \notin d$

- As $p? \notin d$, then $\#(d \cup \{p?\}) = \#d + 1$
  Therefore, $\#d < Max$. As, $\#d < Max \Rightarrow \#d <= Max$
  Hence we can simplify PreEnter1 to:

Investigating Preconditions

PreEnter1

d: P Person
p?: Person

$\#d < Max \wedge$
$p? \notin d$

These two formulae represent the precondition to Enter1

Telephone Book Example:

AddEntry

Members, members': P Person
telephones, telephones': Person $\leftrightarrow$Phone
name?: Person
newnumber?:Person

dom telephones $\subseteq$ members
dom telephones' $\subseteq$ members'
name?$\in$ members
name?   newnumber? $\notin$ telephones
telephones' = telephones $\cup$ { name?   newnumber?}
members' = members

Telephone Book Example:

PreAddEntry

members: P Person
telephones: Person $\leftrightarrow$Phone
name?: Person
newnumber?:Phone

$\exists$members':P Person ; telephones': Person   Phone $\bullet$
(dom telephones $\subseteq$ members $\wedge$
dom telephones' $\subseteq$ members' $\wedge$
name?$\in$ members $\wedge$
name?   newnumber? $\notin$ telephones $\wedge$
telephones' = telephones $\cup$ { name?   newnumber?} $\wedge$
members' = members)

Application of the one point law twice removes $\exists$

$\text{dom telephones} \subseteq \text{members} \wedge$
$\text{dom (telephones} \cup \{ \text{name?} \mapsto \text{newnumber}\}) \subseteq \text{members} \wedge$
$\text{name?} \in \text{members} \wedge$
$\text{name?} \mapsto \text{newnumber?} \notin \text{telephones}$

$\equiv$

$\text{dom telephones} \subseteq \text{members} \wedge$
$\text{name?} \in \text{members} \wedge$
$\text{name?} \mapsto \text{newnumber?} \notin \text{telephones}$

Totality

An operation is **total** if it is defined on every state which satisfies the state invariant

Enter1 is not total. We need to add additional specifications.

Enter1
ΔClass1
p?: Person

$\# d < Max$
$p? \notin d$
$d' = d \cup \{p?\}$

Full1
ΞClass1
p?: Person
rep!:Report

$\# d = Max$
rep! = ‘Class Full’

AlreadyPresent1
ΞClass1
p?: Person
rep!:Report

$p? \in d$
rep! = ‘Already Here’

Success
rep!:Report

rep! = ‘Okay’

$DoEnter1 \triangleq (Enter1 \wedge Success) \vee Full1 \vee AlreadyPresent1$

To show that DoEnter1 is total we calculate PreDoEnter1

PreDoEnter1
d: P Person
p?: Person

$\# d <= Max$

- The schema for DoEnter1 is total because it specifies what happens no matter what values its inputs and before variables have.
- Note, that the values of the before variables have to satisfy the before state invariant

Operation Refinement

- Operation Refinement takes place within a single specification. One one kind of state space is involved.
- It is a way of relating operations defined on the same data.
- The concrete operation OpC is a refinement of the abstract operation OpA, both defined on the same state space, iff both of the following formulae can be proved:
  - pre OpA $\Rightarrow$ pre OpC
    - the concrete operation succeeds whenever the abstract one does
  - pre OpA $\wedge$ OpC $\Rightarrow$ OpA
    - the abstract and the concrete operations produce the same results on the same starting states.