Formal Specification F28FS2, Lecture 6
The rest of Chapter 4, and Chapter 5

Jamie Gabbay

January 27, 2014

Remember

- Propositions: are assigned truth-values.
- Variables: have a type.
- Sets: have elements.
- Schemas: a judgement-form. Pre- and post-conditions. ΔS and ΞS. Input and output variables. Combining schemas. Totalising schemas.

Remember

If S is a schema then S' is the schema written out with primed (dashed) variables. By convention, S represents the universe before (before whatever action we are specifying) and S' the universe after.

ΔS is the pair of S and S' side-by-side with no commitment to any connection between them.

ΞS is a no-op; it puts S and S' side-by-side and asserts that the state is unchanged.

Preconditions

Suppose a schema is of the form

Op
$\Delta State$
morevariables
someconditions

Then *pre Op* is the conditions on *State* and input, and *post Op* is the conditions on $State'$ and output.

If we assign *pre Op* truth-value T then *Op* is total — any state, any output.

Preconditions

pre $(S \vee T)$ is always equal to $(pre\ S) \vee (pre\ T)$ (not a definition; a fact).

$S \setminus x$ is S is x hidden. x is existentially quantified. That means that $S \setminus x$ will give its private copy of x whatever value is necessary to make the spec true.

$\setminus x$ is an abstract form of search. No algorithm — just a search for a suitable x.

Recall: *AddMember*

AddMember

$$\begin{array}{ll} badminton : \mathbb{P}STUDENT, & hall : \mathbb{P}STUDENT \\ badminton' : \mathbb{P}STUDENT, & hall' : \mathbb{P}STUDENT \\ newmember? : STUDENT & \end{array}$$

$$\begin{array}{ll} hall \subseteq badminton & \#hall \leq maxplayers \\ hall' \subseteq badminton' & \#hall' \leq maxplayers' \end{array}$$

$$newmember? \notin badminton$$

$$badminton' = badminton \cup \{newmember?\}$$

$$hall' = hall$$

Recall: *AddMember*

Or more succinctly:

AddMember

Δ *ClubState*

newmember? : STUDENT

$newmember? \notin badminton$

$badminton' = badminton \cup \{newmember?\}$

$hall' = hall$

Recall: *AddMember*

We calculated *pre AddMember* by existentially quantifying (hiding) $\mathsf{badminton}'$ and hall'. So *AddMember* $\setminus \{\mathsf{badminton}', \mathsf{hall}'\}$ seeks some outputs to make the inputs true.

That's what a precondition does: it returns the condition that guarantees that some output and 'after' state exists. We simplified and found that we can find some outputs to make the inputs true, providing that $\mathsf{newmember}? \notin \mathsf{badminton}$.

Recall: *AddMember*

So the operation described by *AddMember* is not defined if newmember$? \in$ badminton.

$$TotalAddMember \mathrel{\widehat{=}} (AddMember \land SuccessMessage) \lor IsMember.$$

IsMember outputs an error message if newmember$? \in$ badminton.

pre TotalAddMember is T.

Totalise *RemoveMember*

RemoveMember

Δ*ClubState*

member? : STUDENT

---

member? $\in$ badminton

badminton$'$ = badminton $\setminus$ {member?}

hall$'$ = hall $\setminus$ {member?}

Precondition: member? $\in$ badminton

Postconditions:

badminton$'$ = badminton $\setminus$ {member?}   hall$'$ = hall $\setminus$ {member?}

Totalising *RemoveMember*

Let MESSAGE ::= success | notMember.

NotMember

Ξ*Clubstate*
member? : STUDENT
outcome! : MESSAGE

---

member? $\notin$ badminton
outcome! = notMember

SuccessMessage

outcome! : MESSAGE

---

outcome! = success

$$TotalRemoveMember \mathrel{\widehat=} (RemoveMember \land SuccessMessage) \lor NotMember$$

Totalising *LeaveHall*

LeaveHall

Δ*ClubState*
leaver? : STUDENT

---

leaver? $\in$ hall
hall$'$ = hall $\setminus$ {leaver}
badminton$'$ = badminton

Precondition: leaver? $\in$ hall.

Totalising *LeaveHall*

$$\mathsf{MESSAGE} ::= \mathsf{success} \mid \mathsf{notInHall}$$

NotInHall

$$\Xi \mathit{Clubstate}$$
$$\mathsf{leaver?} : \mathsf{STUDENT}$$
$$\mathsf{outcome!} : \mathsf{MESSAGE}$$

$$\mathsf{leaver?} \notin \mathsf{hall}$$
$$\mathsf{outcome!} = \mathsf{notInHall}$$

SuccessMessage

$$\mathsf{outcome!} : \mathsf{MESSAGE}$$

$$\mathsf{outcome!} = \mathsf{success}$$

$$\mathit{TotalLeaveHall} \mathrel{\widehat{=}} (\mathit{LeaveHall} \land \mathit{SuccessMessage}) \lor \mathit{NotInHall}$$

Totalising operations with more than one predicate

Our examples so far have only had one precondition, for example:

- $leaver? \in hall$
- $member? \in badminton$
- $newmember? \notin badminton$ (from lecture 5)

Totalising operations with more than one predicate

EnterHall has three preconditions.

EnterHall

$\Delta ClubState$
$enterer? : STUDENT$

$enterer? \in badminton$
$enterer? \notin hall$
$\#hall < maxplayers$
$hall' = hall \cup \{enterer?\}$
$badminton' = badminton$

EnterHall (expanded)

EnterHall

$badminton, hall, badminton', hall' : \mathbb{P} STUDENT,$
$enterer? : STUDENT$

$enterer? \in badminton$
$enterer? \notin hall$
$\#hall < maxplayers$
$hall' = hall \cup \{enterer?\}$
$badminton' = badminton$

EnterHall (hidden)

pre EnterHall

$$badminton, hall : \mathbb{P}STUDENT,$$
$$enterer? : STUDENT$$

$$\exists\, badminton', hall' : \mathbb{P}STUDENT \bullet$$
$$enterer? \in badminton$$
$$\land\ enterer? \notin hall$$
$$\land\ \#hall < maxplayers$$
$$\land\ hall' = hall \cup \{enterer?\}$$
$$\land\ badminton' = badminton$$

EnterHall (hidden, simplified)

pre EnterHall

$\text{badminton}, \text{hall} : \mathbb{P}\text{STUDENT},$
$\text{enterer?} : \text{STUDENT}$

$\text{enterer?} \in \text{badminton}$
$\text{enterer?} \notin \text{hall}$
$\#\text{hall} < \text{maxplayers}$

Unexpectedly easy, really. Bit long, but not too painful.

What about the disappearing logical conjunction ($\wedge$)?

Totalising operations with more than one predicate

Three preconditions:

enterer? $\in$ badminton   enterer? $\notin$ hall   #hall $<$ maxplayers

Don't panic! (What TV series is that from?)

Just write a schema describing what to do if the (several) preconditions are not satisfied, and use disjunction to put them side-by-side with the 'main program' ...

... or ...

... write several schema, one for each precondition.

MESSAGE ::= success | notMember | hallFull | inHall

Exercise 4.5: Totalise *EnterHall*

EnterHall

$\Delta ClubState$
$enterer? : STUDENT$

$enterer? \in badminton$
$enterer? \notin hall$
$\#hall < maxplayers$
$hall' = hall \cup \{enterer?\}$
$badminton' = badminton$

NotMember

$\Xi ClubState$
$enterer? : STUDENT$
$outome! : MESSAGE$

$enterer? \notin badminton$
$outcome! = notMember$

AlreadyInHall

$\Xi ClubState$
$enterer? : STUDENT$
$outome! : MESSAGE$

$enterer? \in hall$
$outcome! = inHall$

HallFull

$\Xi ClubState$
$outome! : MESSAGE$

$\#hall = maxPlayers$
$outcome! = hallFull$

Exercise 4.5: Totalise *EnterHall*

$$\begin{aligned}\mathit{TotalEnterHall} \mathrel{\widehat{=}} & (\mathit{EnterHall} \land \mathit{SuccessMessage}) \\ & \lor \mathit{NotMember} \\ & \lor \mathit{AlreadyInHall} \\ & \lor \mathit{HallFull}\end{aligned}$$

Checking whether an operation is total

$$\begin{aligned} \mathit{TotalEnterHall} \mathrel{\widehat=} & (\mathit{EnterHall} \land \mathit{SuccessMessage}) \lor \mathit{NotMember} \\ & \lor \mathit{AlreadyInHall} \lor \mathit{HallFull} \end{aligned}$$

Is *TotalEnterHall* really total?

To check, calculate *pre TotalEnterHall*.

If this has truth-value T then for all ‘before’ states and inputs, *TotalEnterHall* specifies some ‘after’ state and output — which is what in the language of functions ‘being total’ means.

Checking whether an operation is total

pre distributes over disjunction:

$$\textit{pre TotalEnterHall} \mathrel{\widehat=} \textit{pre } (\textit{EnterHall} \land \textit{SuccessMessage}) \lor \textit{pre NotMember} \lor \textit{pre AlreadyInHall} \lor \textit{pre HallFull}$$

Checking that *TotalEnterHall* is total

You need to be able to check that:

- *pre NotMember* is enterer$? \notin$ badminton.
- *pre AlreadyInHall* is enterer$? \in$ hall.
- *pre HallFull* is hallFull.

But what about *EnterHall* $\wedge$ *SuccessMessage*?

Expand! Hide! Simplify!

$$
\begin{array}{l}
\textit{EnterHall} \wedge \textit{SuccessMessage} \\
\hline
\text{badminton}, \text{hall}, \text{badminton}', \text{hall}' : \mathbb{P}\text{STUDENT}, \\
\text{enterer?} : \text{STUDENT} \\
\text{outcome!} : \text{MESSAGE} \\
\hline
\text{enterer?} \in \text{badminton} \\
\text{enterer?} \notin \text{hall} \\
\#\text{hall} < \text{maxplayers} \\
\text{hall}' = \text{hall} \cup \{\text{enterer?}\} \\
\text{badminton}' = \text{badminton} \\
\text{outcome!} : \text{success} \\
\hline
\end{array}
$$

Expand! Hide! Simplify!

$pre\ (EnterHall \land SuccessMessage)$

$\mathsf{badminton}, \mathsf{hall} : \mathbb{P}\mathsf{STUDENT},$

$\mathsf{enterer?} : \mathsf{STUDENT}$

$\exists\, \mathsf{badminton'}, \mathsf{hall'} : \mathbb{P}\mathsf{STUDENT}, \mathsf{output!} : \mathsf{MESSAGE} \bullet$

$\mathsf{enterer?} \in \mathsf{badminton}$

$\land\ \mathsf{enterer?} \notin \mathsf{hall}$

$\land\ \#\mathsf{hall} < \mathsf{maxplayers}$

$\land\ \mathsf{hall'} = \mathsf{hall} \cup \{\mathsf{enterer?}\}$

$\land\ \mathsf{badminton'} = \mathsf{badminton}$

$\land\ \mathsf{output!} = \mathsf{success}$

Expand! Hide! Simplify!

$$
\begin{array}{l}
\textit{pre}\ (\textit{EnterHall} \land \textit{SuccessMessage}) \\
\hline
\mathsf{badminton}, \mathsf{hall} : \mathbb{P}\mathsf{STUDENT}, \\
\mathsf{enterer}? : \mathsf{STUDENT} \\
\hline
\mathsf{enterer}? \in \mathsf{badminton} \\
\mathsf{enterer}? \notin \mathsf{hall} \\
\#\mathsf{hall} < \mathsf{maxplayers}
\end{array}
$$

That's it; each of these three conditions is covered by the other parts of our disjunction.

Specs education: "Where do Z specifications come from?"

Gee, I'm glad you asked that son. Pop and Mom love specification very very much, and so one day they get together and they do the following:

Specs education time: "Pop . . . where do baby Z specifications come from?"

- Requirements analysis. Identify the sets and constants.
- Identify what variables you want, and what types they'll range over.
- Identify the state schema.
- Identify your initial state, and prove it exists (i.e. some values for the variables can satisfy it; a useful sanity check).
- Identify the operations you want to model.
- Identify the operations' preconditions. Develop error handling schema to handle the cases where those preconditions are not satisfied.
- Totalise the operations.

The badminton club all over again

Basic type: [STUDENT].

Global variable:

$$
\begin{array}{|l}
\text{maxplayers} : \mathbb{N} \\
\hline
\text{maxplayers} = 20
\end{array}
$$

The badminton club all over again

State schema:

ClubState

$badminton : \mathbb{P} STUDENT$
$hall : \mathbb{P} STUDENT$

$hall \subseteq badminton$
$\#hall \leq maxplayers$

The badminton club all over again

Initial state:

InitClubState

$ClubState'$

$badminton' = \{\}$

$hall' = \{\}$

(Recall convention to use 'after' state variables in initial state.)

The badminton club all over again

Preconditions are $\mathsf{hall}' \subseteq \mathsf{badminton}'$ and $\#\mathsf{hall}' \leq \mathsf{maxplayers}$.

$\{\} \subseteq \{\}$ and $0 \leq \mathsf{maxplayers}$ are indeed true.

Operations:

AddMember
(precondition: newMember $\notin$ badminton) (error handler: IsMember)

RemoveMember
(precondition: member $\in$ badminton) (error handler: NotMember)

EnterHall
(precondition: enterer$?$ $\in$ badminton, enterer$?$ $\notin$ hall, $\#$hall $<$ maxPlayers) (error handlers: NotMember$[$enterer$?/$member$?]$, AlreadyInHall, HallFull)

LeaveHall
(precondition: leaver$?$ $\in$ hall) (error handler: NotInHall)

OutsideHall (no preconditions; just a query)

Location (no preconditions; just a query)

Total operators

$$
\begin{aligned}
\mathsf{TotalAddMember} \mathrel{\widehat{=}} & (\mathsf{AddMember} \land \mathsf{SuccessMessage}) \\
& \lor \mathsf{IsMember} \\
\mathsf{TotalRemoveMember} \mathrel{\widehat{=}} & (\mathsf{RemoveMember} \land \mathsf{SuccessMessage}) \\
& \lor \mathsf{NotMember} \\
\mathsf{TotalEnterHall} \mathrel{\widehat{=}} & (\mathsf{EnterHall} \land \mathsf{SuccessMessage}) \\
& \lor \mathit{NotMember} \lor \mathit{AlreadyInHall} \lor \mathit{HallFull} \\
\mathsf{TotalLeaveHall} \mathrel{\widehat{=}} & (\mathsf{LeaveHall} \land \mathsf{SuccessMessage}) \\
& \lor \mathsf{NotInHall}
\end{aligned}
$$

OutsideHall and Location are already total.