Formal Specification F28FS2, Lecture 5

Jamie Gabbay

January 27, 2014

Taking stock

Propositions have truth-values.

Variables have types.

Sets have elements.

Schema assert truths.

If S is a schema then ΔS is a pair of before and after states, with no assertions connecting them, and ΞS is a pair of before and after states, with assertions that they take equal values (think: measurement).

Combining schema

Suppose schema A, B, B':

$$\begin{array}{|l}\hline \textit{A} \\ a : \mathbb{Z} \\ \hline a = 42 \\ \hline\end{array} \qquad \begin{array}{|l}\hline \textit{B} \\ a, b : \mathbb{Z} \\ \hline a = b + 2 \\ b < 10 \\ \hline\end{array} \qquad \begin{array}{|l}\hline \textit{B'} \\ B : \mathbb{P}\mathbb{Z} \\ \hline 42 \in B \\ \hline\end{array}$$

Write $AandB \mathrel{\widehat{=}} A \wedge B$ for the schema which asserts the content of A and B.

$$\begin{array}{|l}\hline \textit{AandB} \\ a, b : \mathbb{Z} \\ \hline a = 42 \wedge (a = (b + 2) \wedge b < 10) \\ \hline\end{array}$$

Combining schema

A and B establish some state variables and predicates on them.
$AandB$ combines these.

Combining schema

Why stop at $\land$? We have $\lor$, $\Rightarrow$, and $\Leftrightarrow$.

The pattern is always the same: combine the state variables and the predicates.

Write $AimpliesB' \mathrel{\widehat{=}} A \Rightarrow B'$ for

$$\begin{array}{|l}
\hline
\textit{AimpliesB}' \\
a : \mathbb{Z},\ B : \mathbb{P}\mathbb{Z} \\
\hline
a = 42 \Rightarrow (42 \in B) \\
\hline
\end{array}$$

We can form $AorB \mathrel{\widehat{=}} A \lor B$.

. . . and so on.

Recall: *ClubState*

ClubState

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

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

This says:

ClubState

- badminton is a set of students (I suppose: the students that play badminton).
- hall is a set of students (the students in the badminton hall, which has a capacity of 20?).
- Students in the hall must play badminton (so they've obviously got a man on the door checking?).
- ... and you can't have more people in the hall than its capacity.

Recall: *AddMember*

AddMember

$$\text{badminton} : \mathbb{P}\text{STUDENT}, \quad \text{hall} : \mathbb{P}\text{STUDENT}$$
$$\text{badminton}' : \mathbb{P}\text{STUDENT}, \quad \text{hall}' : \mathbb{P}\text{STUDENT}$$
$$\text{newmember?} : \text{STUDENT}$$

$$\text{hall} \subseteq \text{badminton} \qquad \#\text{hall} \leq \text{maxplayers}$$
$$\text{hall}' \subseteq \text{badminton}' \qquad \#\text{hall}' \leq \text{maxplayers}'$$
$$\text{newmember?} \notin \text{badminton}$$
$$\text{badminton}' = \text{badminton} \cup \{\text{newmember?}\}$$
$$\text{hall}' = \text{hall}$$

Recall: *AddMember*

Or more succinctly:

AddMember

Δ*ClubState*

$newmember? : STUDENT$

$newmember? \notin badminton$

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

$hall' = hall$

Parenthetic note: Renaming

What if you want to rename variables in a schema?

$S[x/a, y/b, z/c]$ represents S with a renamed to x, b renamed to y, and c renamed to z.

ClubState

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

---

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

FootyClub

football : $\mathbb{P}$STUDENT
pitch : $\mathbb{P}$STUDENT

---

pitch $\subseteq$ football
$\#$pitch $\leq$ maxplayers

$$FootyClub = ClubState[football/badminton, pitch/hall]$$

Recall: *AddMember*

So we can write

$\Delta \mathit{ClubState}$

as

$\mathit{ClubState} \wedge \mathit{ClubState}[\mathrm{hall}'/\mathrm{hall}, \mathrm{badminton}'/\mathrm{badminton}]$.

Refining *AddMember*

hall $\subseteq$ badminton suggests that hall is just the students in the badminton club in the hall.

There may be other people in the hall.

There are the rowers in the corner on their machines, the hockey players, the rock-climbers, maybe even a bit of ping-pong.

If one of these non-badminton-players sees the empty futility of their non-badminton-player ways, they may join the badminton club.

This epiphany might come at any time; while they're in the hall, or even just while they're outside the hall, perhaps studying Formal Spec.

Refining *AddMember*

Introduce an enumerated type LOCATION ::= inside | outside

AddMemberInHall

Δ *ClubState*
newmember? : STUDENT
where? : LOCATION

---

where? = inside
newmember? $\notin$ badminton
#hall < maxPlayers
badminton′ = badminton ∪ {newmember?}
hall′ = hall ∪ newMember?

AddMemberOutHall

Δ *ClubState*
newmember? : STUDENT
where? : LOCATION

---

where? = outside
newmember? $\notin$ badminton
badminton′ = badminton ∪ {newmember?}
hall′ = hall

Refining *AddMember*

$$AddMemberAnywhere \mathrel{\widehat=} AddMemberInHall \lor AddMemberOutHall$$

AddMemberAnywhere describes a program which checks where the member is (inside, ouside) and does the right thing accordingly.

Isn't that a bit magic?

This is a case-split. $\lor$ is a case-split. $\lor$ on schema is a case-split for schema. $\land$ is like a parallel execution.

But there is no notion of flow of control or execution here. Just specifications.

Go on, tell me this isn't pretty. I dare you.

Initial State

What's the initial state of the badminton club?

How about this:

InitClubState

ClubState'

$badminton' = \{\}$
$hall' = \{\}$

It's a convention to use 'after' (with prime; with dash) state variables in initial states.

This is because the initial state takes place after initialisation.

Initial State

The initial state had better satisfy the conditions for *ClubState*.

That is, $\text{hall}' \subseteq \text{badminton}'$ and $\#\text{hall}' \leq \text{maxplayers}$.

So let's check $\{\} \subseteq \{\}$ and $0 \leq \text{maxplayers}$.

Totalising operations

$$
\begin{array}{|l}
\hline
\textit{AddMember} \\
\Delta \textit{ClubState} \\
\text{newmember?} : \text{STUDENT} \\
\hline
\text{newmember?} \notin \text{badminton} \\
\text{badminton}' = \text{badminton} \cup \{\text{newmember?}\} \\
\text{hall}' = \text{hall} \\
\hline
\end{array}
$$

Note the precondition $\text{newmember?} \notin \text{badminton}$.

What if not $\text{newmember?} \notin \text{badminton}$. (So $\text{newmember?} \in \text{badminton}$ holds.)

Not *Addmember*'s problem: *AddMember* specifies the behaviour of a PARTIAL function.

Totalising operations

What do we do about this in Z? How do we make this specification of a partial function, into a specification of a total function?

We need to totalise the schema.

Totalising operations

Recall the no-op:

$\Xi ClubState$

$\Delta ClubState$

$badminton' = badminton$
$hall' = hall$

$\Xi ClubState$

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

$hall \subseteq badminton, \ \#hall \leq maxplayers$
$hall' \subseteq badminton', \ \#hall' \leq maxplayers'$
$hall' = hall, \ badminton' = badminton$

Totalising operations

$$\mathrm{MESSAGE} ::= \mathrm{success} \mid \mathrm{isMember}$$

IsMember

$\Xi \mathit{ClubState}$
$\mathrm{newMember?} : \mathrm{STUDENT}$
$\mathrm{outcome!} : \mathrm{MESSAGE}$

$\mathrm{newMember?} \in \mathrm{badminton}$
$\mathrm{outcome!} = \mathrm{isMember}$

SuccessMessage

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

$\mathrm{outcome!} = \mathrm{SUCCESS}$

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

Totalising operations

Programs in C and Java are automatically total; they take an input, give an output.

A schema is total when the outcome is specified for all possible inputs. Schema can be partial.

Go through the previous specs: *RemoveMember*, *EnterHall*, *LeaveHall*, *NotInHall*. Which of these are total? Totalise the ones that are not.

Hiding

$S \setminus b$ is the schema obtained by existentially quantifying b in S.
Best explained by example:

A

$a : \mathbb{Z}$

$a = 42$

HideA

$\exists\, a : \mathbb{Z} \bullet$
$a = 42$

B

$a, b : \mathbb{Z}$

$a = b + 2$
$b < 10$

HideB

$a : \mathbb{Z}$

$\exists\, b : \mathbb{Z} \bullet$
$(a = b + 2 \land$
$b < 10)$

Similarly for $S \setminus a, b$ and so on.

Hiding

Note that $\exists\, b : \mathbb{Z} \bullet (a = b + 2 \land b < 10)$ means the same thing as $a < 12$.

So we can equivalently write *HideB* as:

$$\begin{array}{|l}\hline \textit{HideB} \\ a : \mathbb{Z} \\ \hline a < 12 \\ \hline \end{array}$$

Another example of hiding

Define $AddWho \mathrel{\widehat=} AddMember \setminus \text{newMember?}$:

AddMember

$\Delta ClubState$
$\text{newmember?} : \text{STUDENT}$

$\text{newmember?} \notin \text{badminton}$
$\text{badminton}' = \text{badminton} \cup \{\text{newmember?}\}$
$\text{hall}' = \text{hall}$

AddWho

$\Delta ClubState$

$\exists \text{newmember?} : \text{STUDENT} \bullet$
$(\text{newmember?} \notin \text{badminton} \land$
$\text{badminton}' = \text{badminton} \cup \{\text{newmember?}\} \land$
$\text{hall}' = \text{hall})$

Calculating preconditions

Define $\mathit{SuccessAddMember} \mathrel{\widehat{=}} \mathit{AddMember} \land \mathit{SuccessMessage}$.

AddMember

$\Delta \mathit{ClubState}$
$\mathrm{newmember?} : \mathrm{STUDENT}$

$\mathrm{newmember?} \notin \mathrm{badminton}$
$\mathrm{badminton'} = \mathrm{badminton} \cup \{\mathrm{newmember?}\}$
$\mathrm{hall'} = \mathrm{hall}$

SuccessMessage

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

$\mathrm{outcome!} = \mathrm{SUCCESS}$

I bet you don't understand that. It's got a bit complicated, hasn't it?

Partially expand the definition

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{SuccessAddMember}} \\
\hline
\Delta \textit{ClubState} \\
\text{newmember?} : \text{STUDENT} \\
\text{outcome!} : \text{MESSAGE} \\
\hline
\text{newmember?} \notin \text{badminton} \\
\text{badminton}' = \text{badminton} \cup \{\text{newmember?}\} \\
\text{hall}' = \text{hall} \\
\text{outcome!} = \text{success} \\
\hline
\end{array}
$$

That's a bit better — but not good enough. We want to expand more!

Expand further

SuccessAddMember

ClubState

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

$newmember? : STUDENT$

$outcome! : MESSAGE$

$hall' \subseteq badminton'$

$\# hall' \leq maxPlayers$

$newmember? \notin badminton$

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

$hall' = hall$

$outcome! = success$

Calculating preconditions

Recall:

$\mathrm{badminton}', \mathrm{hall}' : \mathbb{P}\mathrm{STUDENT}$ are the state after.

$\mathrm{badminton}, \mathrm{hall} : \mathbb{P}\mathrm{STUDENT}$ are the state before.

newMember? is the input.

output! is the output.

It is Z convention to so name variables: $'$ for after, ? for input, ! for output.

AddMemberSuccess is an (abstract) program, just like in a real programming language.

But is it defined for all input states and all inputs?

$SuccessAddMember \setminus \{\mathsf{badminton}', \mathsf{hall}', \mathsf{output!}\}$

pre SuccessAddMember

ClubState

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

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

$\mathsf{hall}' \subseteq \mathsf{badminton}' \wedge \#\mathsf{hall}' \leq \mathsf{maxPlayers}$

$\wedge\ \mathsf{newmember?} \notin \mathsf{badminton}$

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

$\wedge\ \mathsf{hall}' = \mathsf{hall} \wedge \mathsf{outcome!} = \mathsf{success}$

Set $\mathsf{hall}' = \mathsf{hall}$ and drop outcome!.

$\exists\, \mathsf{outcome!} : \mathsf{MESSAGE} \bullet \mathsf{outcome!} = \mathsf{success}$ is true and we do not mention outcome! elsewhere.

SuccessAddMember $\setminus \{badminton', hall', output!\}$, simplified

pre SuccessAddMember

ClubState
newmember? : STUDENT

$\exists$ badminton′ $\bullet$
hall $\subseteq$ badminton′ $\wedge$ #hall $\leq$ maxPlayers
$\wedge$ newmember? $\notin$ badminton
$\wedge$ badminton′ $=$ badminton $\cup$ {newmember?}
$\wedge$ hall $=$ hall

We drop hall $=$ hall and note that #hall $\leq$ maxPlayers, which was a condition on hall′, is now something that's already in *ClubState*.

$\mathit{SuccessAddMember} \setminus \{\text{badminton}', \text{hall}', \text{output!}\}$, simplified more

pre SuccessAddMember

ClubState

newmember? : STUDENT

$\exists\, \text{badminton}' \bullet$

$\text{hall} \subseteq \text{badminton}' \land \text{newmember?} \notin \text{badminton}$

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

hall $\subseteq$ badminton by *ClubState* and $\text{badminton}' = \text{badminton} \cup \{\text{newmember?}\}$, so $\text{hall} \subseteq \text{badminton}'$ is guaranteed.

SuccessAddMember $\setminus$ {badminton′, hall′, output!}, simplified even more

pre SuccessAddMember
ClubState
newmember? : STUDENT
$\exists$ badminton′ •
newmember? $\notin$ badminton
$\wedge$ badminton′ = badminton $\cup$ {newmember?}

$\exists$ badminton′ • badminton′ = badminton $\cup$ {newmember?} is as useful as a barber shop on the steps of the guillotine; cut it off.

SuccessAddMember $\setminus$ {badminton′, hall′, output!}, simplified ridiculously

pre SuccessAddMember
ClubState
newmember? : STUDENT
newmember? $\notin$ badminton

There's your precondition: newmember? $\notin$ badminton.

We found a but. The program fails if newmember? $\in$ badminton.

pre SuccessAddMember

Another description is this:

The operation described by *SuccessAddMember* is not total; it is not defined if newmember$?$ $\in$ badminton.

Fact

Fact. *pre* distributes over disjunction:

$$pre\ (S \vee T) = pre\ S \vee pre\ T.$$

So to check if *TotalAddMember* really is total, it suffices to calculate *pre IsMember* and see if it is newMember? $\in$ badminton.

Let's do it: let our slogan be expand, hide, simplify.

Expand, hide, simplify: IsMember

IsMember

$\Xi ClubState$
newMember? : STUDENT
outcome! : MESSAGE

---

newMember? $\in$ badminton
outcome! $=$ isMember

IsMember

ClubState
badminton$'$, hall$'$: $\mathbb{P}$STUDENT
newMember? : STUDENT
outcome! : MESSAGE

---

hall$'$ $\subseteq$ badminton$'$
$\#$hall$'$ $\leq$ maxPlayers
newMember? $\in$ badminton
outcome! $=$ isMember
badminton$'$ $=$ badminton
hall$'$ $=$ hall

Expand, hide, simplify: IsMember

pre IsMember

$ClubState$

$newMember? : STUDENT$

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

$hall' \subseteq badminton'$

$\wedge\ \#hall' \leq maxPlayers$

$\wedge\ newMember? \in badminton$

$\wedge\ outcome! = isMember$

$\wedge\ badminton' = badminton$

$\wedge\ hall' = hall$

Expand, hide, simplify: IsMember

pre IsMember

ClubState

newMember? : STUDENT

$\exists$ outcome! : MESSAGE$\bullet$

hall $\subseteq$ badminton

$\wedge$ #hall $\leq$ maxPlayers

$\wedge$ newMember? $\in$ badminton

$\wedge$ outcome! $=$ isMember

(Don't rush this. One step at a time.)

Expand, hide, simplify: IsMember

pre IsMember

ClubState

newMember? : STUDENT

$hall \subseteq badminton$

$\land \#hall \leq maxPlayers$

$\land newMember? \in badminton$

Expand, hide, simplify: IsMember

$$
\begin{array}{|l}
\hline
\textit{pre IsMember} \\
\textit{ClubState} \\
\text{newMember?} : \text{STUDENT} \\
\hline
\text{newMember?} \in \text{badminton} \\
\hline
\end{array}
$$

That's it, we're done. *TotalAddMember* is total.

$$
\begin{aligned}
\textit{pre TotalAddMember} &= \textit{pre SuccessAddMember} \lor \textit{pre IsMember} \\
&= \text{newMember?} \notin \text{badminton} \lor \text{newMember?} \in \text{badminton} \\
&= T
\end{aligned}
$$