FROM Z TO PVS—VIA THE ACCOUNTING SYSTEM EXAMPLE

STEVE REEVES

1. Aims

To show how, given a Z specification, we can turn it into a PVS specification, and then prove things.

Done via a simple accounting system as an example.

2. A Z specification of a simple accounting system

We have given sets

$[DATE, TEXT, MONEY]$

and the useful schema

Transaction
$date : DATE$
$description : TEXT$
$amount : MONEY$

Now the state space for the account is just a sequence of transactions (with an initial state with no transactions):

SingleState
$ssaccount : \mathrm{seq}\ Transaction$

InitSingleState
$SingleState$

$saccount = <>$

Then we finally have a single state-changing operation, which itself uses an operation that just sets-up a new transaction (this follows a, simplified, version of the Z idiom of promotion...done here just because this might be a useful bit of structuring when things get more complicated later..)

$$
\begin{array}{|l}
\hline
\textit{NewTransaction} \\
\textit{Transaction}' \\
\textit{date}? : \textit{DATE} \\
\textit{description}? : \textit{TEXT} \\
\textit{amount}? : \textit{MONEY} \\
\hline
\textit{date}' = \textit{date}? \\
\textit{description}' = \textit{description}? \\
\textit{amount}' = \textit{amount}? \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
\Phi \textit{DoSomeBusiness} \\
\Delta \textit{SingleState} \\
\textit{Transaction}' \\
\hline
\textit{saccount}' = \textit{saccount} ^\frown < \theta \textit{Transaction}' > \\
\hline
\end{array}
$$

We then promote this local operation (which works on a single transaction) to the whole system ("the accounts") using the following:

$$\textit{DoSomeBusiness}_0 \mathrel{\widehat=} \exists\, \textit{Transaction}' \bullet \Phi \textit{DoSomeBusiness} \wedge \textit{NewTransaction}$$

3. Moving from the Z to a PVS rendition step by step

The basic simple correspondence between Z schema types (like *Transaction*) and PVS record types is the starting point. We can then go on to deal with schemas themselves since they are just schema types extended to be dependent, in the sense that they use a predicate, which depends on the observations in the schema, to pick out only *some* bindings to be in the schema (considered as a set) rather than *all* bindings in the type, which is what schema types do, i.e. schemas are just schema types with (perhaps) some bindings excluded due to the predicate part.

As we see, this gets very messy eventually because PVS does not have counterparts to the central Z idea of treating sets and types as the same thing. Nor does it have inclusions or any of the convenient language features that gives a *schema calculus* in Z (as far as I can tell). But, the aim is simply to allow a Z specification to be transliterated into correct (if messy) PVS so that proofs etc. might be carried out. No one needs to read it! (It turns out not to be so bad, in fact.)

Even before that, though, we have some given sets to deal with. Since these are very abstract (they come with no information save that they exist) then they act as what PVS would consider to be type constants (and later we might be more sophisticated and model them as parameters for a theory).

For now, then, we have:

$$[\textit{DATE}, \textit{TEXT}, \textit{MONEY}]$$

is given in PVS as

$$\textit{DATE}, \textit{TEXT}, \textit{MONEY} : \textit{TYPE}$$

Then we have (since we are dealing with just a schema type, i.e. no predicate part):

Transaction
$date : DATE$
$description : TEXT$
$amount : MONEY$

is given in PVS as

$TransactionType : TYPE = [\#\ date : DATE, description : TEXT, amount : MONEY\ \#]$

and the same idea works for the next schema type:

SingleState
$ssaccount : \mathrm{seq}\ Transaction$

which in PVS is

$SingleStateType : TYPE = [\#\ ssaccount : finseq[TransactionType]\ \#]$

When we move to more general schemas, i.e. not just schema types, then we are, in PVS terms, dealing with dependency—a value is in the type if some conditions *on the value* are met, so whether the value is in the type or not depends on whether the condition on the value is met or not. Since the values in question come from a schema type (ultimately) then they are bindings (in Z) or records (in PVS), which we saw at the level of types in *Transaction* and *SingleState* above. In Z terms we view a schema as a set of bindings whose elements satisfy the predicate part of the schema in question. So, in PVS, we deal with a set of records whose components satisfy a predicate. We have:

InitSingleState
$SingleStateType$

$saccount = \langle\rangle$

rendered in PVS as

$$InitSingleStateSchema : setof[SingleStateType] = \{s : SingleStateType \mid s`ssaccount = empty\_seq\}$$

However, note that whereas *SingleState* is a PVS type, *InitSIngleStateSchema* is a PVS set, and PVS insists that these are different things (unlike Z, which insists they are the same things). Later, we might want the use the duality that Z affords us, and to handle this we have to introduce a new value

$InitSingleStateType : TYPE = \{s : SingleStateType \mid TRUE\}$

This is somewhat confusing as PVS uses the same notation for a "real" set like *InitSingleStateSchema* which contains bindings, and a type like *InitSingleStateType* which is the type of such bindings. Of course, this is just setting up a synonym for *SingleStateType* in this trivial case.

We continue, recalling that so-called "operation schemas" in Z are just schemas which follow the convention that (in general) their bindings (and hence their types) consist of observations from a "pre-state", observations from a "post-state" which are primed, and observations of any inputs to and outputs from the operation, indicated by having the observation names end in “?” or “!” respectively. Here we follow that convention as far as PVS allows. In the first example here there are no pre-state observations, and sadly PVS does not allow “primes” in identifiers (there seems to have been a worldwide demand for this, and I’m told things are changing in version 7) so we use underscore instead (and question marks ARE allowed, but later we will see that exclamation marks are not), or inclusions as such (hence the use of WITH to keep at least some helpful structuring) and even with the WITH we have to introduce a new type identifier since PVS does not allow type expressions where a type identifier is allowed, which is odd (again I’m told this is not so, but an error in the current compiler rules this out....) :

$NewTransaction$

$$\begin{array}{l} Transaction' \\ date? : DATE \\ description? : TEXT \\ amount? : MONEY \\ \hline date' = date? \\ description' = description? \\ amount' = amount? \end{array}$$

in PVS is

$$TransactionType\_ : TYPE = [\#date\_ : DATE, description\_ : TEXT, amount\_ : MONEY\#]$$

$$\begin{aligned} NTType : TYPE = TransactionType\_\ WITH\ [\#&date? : DATE, \\ &description? : TEXT, \\ &amount? : MONEY\#] \end{aligned}$$

$$\begin{aligned} NewTransactionSchema : setof[NTType] = \{nt : NTType \mid\ &nt`date\_ = nt`date?\ AND \\ &nt`description\_ = nt`description?\ AND \\ &nt`amount\_ = nt`amount?\} \end{aligned}$$

Now consider

$\Phi DoSomeBusiness$

$$\begin{array}{l} \Delta SingleState \\ Transaction' \\ \hline saccount' = saccount^\frown < \theta Transaction' > \end{array}$$

The Δ-inclusion stands for two inclusions of $SingleState$ and $SingleState'$ so we have to define this. We have the following:

$$SingleStateType\_ : TYPE = [\#\ ssaccount\_ : finseq[TransactionType]\ \#]$$

$$DeltaSingleStateType : TYPE = SingleStateType\ \ WITH\ \ SingleStateType\_$$

$$PhiDSBType : TYPE = DeltaSingleStateType\ \ WITH\ \ TransactionType\_$$

$$\begin{aligned}
&PhiDoSomeBusinessSchema : setof[PhiDSBType] = \\
&\qquad \{phidsb : PhiDSBType \mid phidsb`ssaccount_ = \\
&\qquad\qquad o(phidsb`ssaccount, \\
&\qquad\qquad description := phidsb`description_, \\
&\qquad\qquad amount := phidsb`amount_\#)))\}
\end{aligned}$$

(where *singfs* turns a value into the single item in a singleton finite sequence).

Now, hiding of types in PVS.....!!!!! It's already apparent that there's no counterpart to the schema calculus in PVS (which is OK since there are no schemas :)) so hiding (existential quantification over schemas) needs to be modelled inside the set brackets in the predicate part... There is a certain amount of regularity to this, given the above, though...but first we need to unfold the existentially quantified schema expression...so the work we did above concerning *PhiDSB* and so on was, in the end, wasted since we cannot in PVS use it...

$DoSomeBusiness \mathrel{\widehat{=}} \exists\, Transaction' \bullet \Phi DoSomeBusiness \land NewTransaction$

This is an equal, alternative definition:

$$\begin{aligned}
&\textit{DoSomeBusiness} \\
&\Delta SingleState \\
&date? : DATE \\
&description? : TEXT \\
&amount? : MONEY \\
\hline
&\exists\, d : DATE;\ desc : DESCRIPTION;\ a : MONEY \bullet \\
&\qquad d = date? \land desc = description? \land a = amount? \land \\
&\qquad saccount' = saccount \frown \langle\!| date \mapsto d, description \mapsto desc, amount \mapsto a |\!\rangle
\end{aligned}$$

as some schema calculating shows.

This can, using the ideas we have already seen, be directly modelled as:

$$\begin{aligned}
&DSBType : TYPE = DeltaSingleState\ \ WITH\ [\#date? : DATE, \\
&\qquad description? : TEXT, \\
&\qquad amount? : MONEY\#]
\end{aligned}$$

$$\begin{aligned}
&DoSomeBusinessSchema : setof[DSBType] = \\
&\qquad \{dsb : DSB_Type \mid EXISTS(d : DATE, desc : TEXT, a : MONEY) : \\
&\qquad\qquad d = dsb`date?\ AND\ desc = dsb`description?\ AND \\
&\qquad\qquad a = dsb`amount?\ AND \\
&\qquad\qquad dsb`ssaccount\_ = o(dsb`ssaccount, singfs((\#date := d, \\
&\qquad\qquad description := desc, \\
&\qquad\qquad amount := a\#)))\}
\end{aligned}$$

BUT, on second thoughts, if we equalise the types of the schemas we are including in a new schema (*PhiDoSomeBusiness* and *NewTransaction* in this case) then we can use set union to get the new schema over the enlarged (and now common to the two component schemas) type. So an alternative to the above goes as follows.

PhiDoSomeBusinessSchema has its type expanded to the type of *DoSomeBusinessSchema* by defining it as:

$$
\begin{aligned}
&PhiDoSomeBusinessSchema : setof[DSBType] = \\
&\qquad \{phidsb : DSBType \mid phidsb`ssaccount\_ = \\
&\qquad\qquad o(phidsb`ssaccount, \\
&\qquad\qquad description := phidsb`description\_, \\
&\qquad\qquad amount := phidsb`amount\_\#)))\}
\end{aligned}
$$

4. Preconditions

We can model this by first removing components from the type used in the declaration part of the set that represents a schema (mirroring the removal of declarations from the declaration part of a schema), and then (again mirroring what happens to a schema) existentially quantifying the predicate part of the set so as to bind (and hide by abstraction) the just-removed components (observations). This is actually just what we did in the previous example, of course, since that was about hiding too.

An example: given

$$
\begin{array}{|l}
\textit{DoSomeBusiness} \\
\hline
\Delta SingleState \\
date? : DATE \\
description? : TEXT \\
amount? : MONEY \\
\hline
\exists\, d : DATE;\ desc : DESCRIPTION;\ a : MONEY \bullet \\
\quad d = date? \land desc = description? \land a = amount? \land \\
\quad saccount' = saccount \frown \langle\!|date \mapsto d, description \mapsto desc, amount \mapsto a|\!\rangle \\
\hline
\end{array}
$$

we want its precondition. This means hiding all the primed observations in its declaration part (which means hiding $SingleState'$). Recall that the pVS set modelling this schema is:

$$
\begin{aligned}
&DoSomeBusinessSchema : setof[DSBType] = \\
&\qquad \{dsb : DSB\_Type \mid EXISTS(d : DATE, desc : TEXT, a : MONEY) : \\
&\qquad\quad d = dsb`date?\ AND\ desc = dsb`description?\ AND \\
&\qquad\quad a = dsb`amount?\ AND \\
&\qquad\quad dsb`ssaccount\_ = o(dsb`ssaccount, singfs((\#date := d, \\
&\qquad\quad description := desc, \\
&\qquad\quad amount := a\#)))\}
\end{aligned}
$$

so we first need to remove the $SingleState'$ observations from $DSBType$, which gives:

$$preDSBType : TYPE = SingleState$$

and the use this in the declaration part of the new modelling set, together with existentially quantifying (renamed, for clarity) the observations (components) just removed:

$$
\begin{aligned}
&preDoSomeBusinessSchema : setof[preDSBType] = \\
&\qquad \{pdsb : preDSBType \mid EXISTS(s : finiteseq[Transaction]) : \\
&\qquad\quad EXISTS(d : DATE, desc : TEXT, a : MONEY) : \\
&\qquad\qquad d = pdsb`date?\ AND\ desc = pdsb`description?\ AND \\
&\qquad\qquad a = pdsb`amount?\ AND \\
&\qquad\qquad s = o(pdsb`ssaccount, singfs((\#date := d, \\
&\qquad\qquad\quad description := desc, \\
&\qquad\qquad\quad amount := a\#)))\}
\end{aligned}
$$

5. Now for some proofs...