MODULE *NewLinearSnapshotPS*

This module adds a prophecy variable and then a stuttering variable to specification *Spec* of module *NewLinearSnapshot*. It then shows that the resulting spec *SpecPS* implements the linearizable specification *Spec* of module *LinearSnapshot*. This shows that if *Spec_NL* is specification *Spec* of *NewLinearSnaphot* and *Spec_L* is specification *Spec* of *LinearSnapshot*, then $\boldsymbol{\exists}\, mem, rstate, wstate : Spec\_NL$ implies $\boldsymbol{\exists}\, mem, istate : Spec\_L$.

EXTENDS *NewLinearSnapshot*

We first add the prophecy variable p so that, whenever a reader i is executing a read operation, $p[i]$ predicts the value j such that the $EndRd(i)$ action will produce $rstate[i][j]$ as its output.

Our definitions make use of the operators defined in module *Prophecy*. You should read that module to understand the meanings of those operators. We begin by defining the set *Pi* of possible predictions and the domain *Dom* of p.

$$Pi \triangleq Nat \setminus \{0\}$$
$$Dom \triangleq \{r \in Readers : rstate[r] \neq \langle\rangle\}$$

INSTANCE *Prophecy* WITH $DomPrime \leftarrow Dom'$

To add the prophecy variable, it's convenient to use a different disjunctive representation of the next-state action *Next* of *Spec* than we used in writing its definition. Instead of having $EndRd(i)$ as a subaction, we want to write it as an existential quantification over the action $IEndRd(i, j)$, which is an $EndRd(i)$ action that returns $rstate[i][j]$ as its output. We now define *IEndRd* and define *Nxt* to the the next-state action *Next* rewritten in terms of *IEndRd*. Of course, *Nxt* should be equivalent to *Next*.

$$\begin{aligned}
IEndRd(i, j) \triangleq\ & \land interface[i] = NotMemVal \\
& \land interface' = [interface \text{ EXCEPT } ![i] = rstate[i][j]] \\
& \land rstate' = [rstate \text{ EXCEPT } ![i] = \langle\rangle] \\
& \land \text{UNCHANGED } \langle mem, wstate\rangle
\end{aligned}$$

$$\begin{aligned}
Nxt \triangleq\ & \lor \exists i \in Readers : \lor BeginRd(i) \\
& \qquad\qquad\qquad\quad \lor \exists j \in 1 \,..\, Len(rstate[i]) : IEndRd(i, j) \\
& \lor \exists i \in Writers : \lor \exists cmd \in RegVals : BeginWr(i, cmd) \\
& \qquad\qquad\qquad\quad \lor DoWr(i) \lor EndWr(i)
\end{aligned}$$

We should check that we haven't made a mistake, and that *Nxt* is indeed equivalent to *Next*. We could use *TLC* to check this. However, the equivalence is so obvious that the *TLAPS* proof system can check it easily by just expanding the appropriate definitions. So, we let *TLAPS* do the checking of the following theorem.

THEOREM $Next \equiv Nxt$

BY DEF *Next*, *Nxt*, *EndRd*, *IEndRd*

Adding the prophecy variable requires replacing each subaction A of the disjunctive representation implied by the definition of *Nxt* with an action Ap. Each action Ap is defined by defining:

- An operator *PredA*, where $PredA(p)$ is the prediction that the value of p is making about action A .

– *PredDomA*, the subset of *Dom* consisting of the elements d for which $p[d]$ is used in the prediction $PredA(p)$.

– *DomInjA*, an injection from a subset of *Dom* to *Dom'* describing the correspondence between predictions made by p and those made by p'. For the prophecy variable we are defining, *DomInjA* is the identity function on $Dom \cap Dom'$. The *Prophecy* module defines $IdFcn(S)$ to be the identity function on the set S.

These definitions for each subaction A follow. For example, *PredBeginRd* is *PredA* for A the $BeginRd(i)$ action.

$$
\begin{aligned}
&PredBeginRd(p) &&\triangleq \text{TRUE}\\
&PredDomBeginRd &&\triangleq \{\}\\
&DomInjBeginRd &&\triangleq IdFcn(Dom)
\end{aligned}
$$

For the $IEndRd(p, i, j)$ action, the *PredA* operator depends on the values of the two bound identifiers in the action's context (named i and j in the definition of *Nxt*).

$$
\begin{aligned}
&PredIEndRd(p, i, j) &&\triangleq j = p[i]\\
&PredDomIEndRd(i) &&\triangleq \{i\}\\
&DomInjIEndRd &&\triangleq IdFcn(Dom')
\end{aligned}
$$

$$
\begin{aligned}
&PredBeginWr(p) &&\triangleq \text{TRUE}\\
&PredDomBeginWr &&\triangleq \{\}\\
&DomInjBeginWr &&\triangleq IdFcn(Dom)
\end{aligned}
$$

$$
\begin{aligned}
&PredDoWr(p) &&\triangleq \text{TRUE}\\
&PredDomDoWr &&\triangleq \{\}\\
&DomInjDoWr &&\triangleq IdFcn(Dom)
\end{aligned}
$$

$$
\begin{aligned}
&PredEndWr(p) &&\triangleq \text{TRUE}\\
&PredDomEndWr &&\triangleq \{\}\\
&DomInjEndWr &&\triangleq IdFcn(Dom)
\end{aligned}
$$

We next define the formula *Condition*, where the property

$Spec \Rightarrow Condition$

must hold for the specification *SpecP* defined below be obtained from *Spec* by adding an auxiliary variable–that is, for *SpecP* to satisfy:

$(\exists\, p : SpecP) \equiv Spec$

Note how the LAMBDA expression is used to "convert" *PredIEndRd* to the single-argument operator with the appropriate meaning required as an argument to *PropohCondition*.

$$
\begin{aligned}
&Condition \triangleq\\
&\Box[\land \forall i \in Readers :\\
&\qquad\quad \land ProphCondition(BeginRd(i), DomInjBeginRd,\\
&\qquad\qquad\qquad\qquad PredDomBeginRd, PredBeginRd)\\
&\qquad\quad \land \forall j \in 1 \,..\, Len(rstate[i]) :\\
&\qquad\qquad ProphCondition(IEndRd(i, j), DomInjIEndRd,\\
&\qquad\qquad\qquad\qquad PredDomIEndRd(i),\\
&\qquad\qquad\qquad\qquad \text{LAMBDA } p : PredIEndRd(p, i, j))\\
&\quad \land \forall i \in Writers :\\
&\qquad\quad \land \forall cmd \in RegVals :
\end{aligned}
$$

$$
\begin{aligned}
&\quad PrSophCondition(BeginWr(i, cmd), DomInjBeginWr, \\
&\qquad\qquad PredDomBeginWr, PredBeginWr) \\
&\quad \land PrSophCondition(DoWr(i), DomInjDoWr, PredDomDoWr, \\
&\qquad\qquad PredDoWr) \\
&\quad \land PrSophCondition(EndWr(i), DomInjEndWr, PredDomEndWr, \\
&\qquad\qquad PredEndWr) \\
&]_{vars}
\end{aligned}
$$

You can have TLC check the property $Spec \Rightarrow Condition$ by temporarily ending the module here and creating a model with $Spec$ as the specification and $Condition$ as a property to be checked.

We now declare the variable p and define $SpecP$ using the $PrSophAction$ operator defined in the $Prophecy$ module. That operator defines

$$PrSophAction(A, p, p', DomInjA, PredDomA, PredA)$$

to be the action Ap that replaces the subaction A . The general form of the initial predicate is $Init \land (p \in [Dom \rightarrow Pi])$. Since $Init$ implies that Dom is the empty set, $[Dom \rightarrow Pi]$ is just $\{EmptyFcn\}$, where $EmptyFcn$ is defined in the $Prophecy$ module to equal the function with empty domain, and $p \in \{EmptyFcn\}$ of course is equivalent to $p = EmptyFcn$.

Note how $SpecP$ is the conjunction of $InitP \land \Box[NextP]\_varsP$ and the liveness property of $Spec$.

VARIABLE p

$$varsP \triangleq \langle vars, p \rangle$$

$$TypeOKP \triangleq TypeOK \land (p \in [Dom \rightarrow Pi])$$

$$InitP \triangleq Init \land (p = EmptyFcn)$$

$$
\begin{aligned}
BeginRdP(i) \triangleq\ & PrSophAction(BeginRd(i), p, p', DomInjBeginRd, \\
& \qquad PredDomBeginRd, PredBeginRd)
\end{aligned}
$$

$$
\begin{aligned}
BeginWrP(i, cmd) \triangleq\ & PrSophAction(BeginWr(i, cmd), p, p', DomInjBeginWr, \\
& \qquad PredDomBeginWr, PredBeginWr)
\end{aligned}
$$

$$
\begin{aligned}
DoWrP(i) \triangleq\ & PrSophAction(DoWr(i), p, p', DomInjDoWr, PredDomDoWr, \\
& \qquad PredDoWr)
\end{aligned}
$$

$$
\begin{aligned}
& IEndRdP(i, j) \triangleq \\
& \quad PrSophAction(IEndRd(i, j), \ p, p', DomInjIEndRd, \\
& \qquad\qquad PredDomIEndRd(i), \text{LAMBDA } q : PredIEndRd(q, i, j))
\end{aligned}
$$

$$
\begin{aligned}
EndWrP(i) \triangleq\ & PrSophAction(EndWr(i), p, p', DomInjEndWr, PredDomEndWr, \\
& \qquad PredEndWr)
\end{aligned}
$$

$$
\begin{aligned}
NextP \triangleq\ & \lor \exists i \in Readers : \lor BeginRdP(i) \\
& \qquad\qquad\qquad\quad \lor \exists j \in 1 \,..\, Len(rstate[i]) : IEndRdP(i, j) \\
& \lor \exists i \in Writers : \lor \exists cmd \in RegVals : BeginWrP(i, cmd) \\
& \qquad\qquad\qquad\quad \lor DoWrP(i) \lor EndWrP(i)
\end{aligned}
$$

$$SpecP \triangleq InitP \land \Box[NextP]_{varsP} \land Fairness$$

We now define *SpecPS* by adding to *SpecP* a stuttering variable s that adds:

- A single stuttering step immediately after a $BeginRdP(i)$ step if $p[i]$ predicts that $EndRdP(i)$ will produce as output $rstate[1]$.

– *Stuttering* steps immediately after $DoWrP(i)$, one such step for every reader j for which $p[j]$ predicts that the $EndRdP(j)$ step will produce as output the value of *mem* immediately after the $DoWrP(i)$ step.

Each of these stuttering steps will become a *DoRd* step of *LinearSnapshot* under our refinement mapping.

A stuttering variable s is added by replacing each subaction A of a disjunctive decomposition of the next-state action by an action As . Action As adds stuttering steps to A by setting the component $s.val$ to an initial value *InitVal* and having each stuttering step replace $s.val$ by $decr(s.val)$ for a "decrementing" operator *decr*, until $s.val$ has the value *bot* . To add a single stuttering step to $BeginRd(i)$, we let $InitVal = 1$, $bot = 0$, and $decr(x) = x - 1$. To add a stuttering step after $DoWr(i)$ for each reader in a set R of readers, we let $InitVal = R$, $bot = \{\}$, and $decr(x)$ equal a set obtained by removing a single element from the set x.

The correctness condition for adding stuttering steps to an action A is that there is some set *Sigma* such that (1) $InitVal \in Sigma$, (2) $bot \in Sigma$, and (3) for any $sig \in Sigma$, repeatedly decrementing *sig* with *decr* eventually reaches *bot* . For $BeginRd(i)$ we let *Sigma* equal $\{0, 1\}$; for $DoWr(i)$ we let *Sigma* equal the set SUBSET *Readers* of all subsets of the set *Readers*. The two theorems below express condition (1) for these two subactions. They are trivially true because the following two formulas are trivially true:

$(\text{IF} \ \ldots \ \text{THEN } 1 \ \text{ELSE} \ 0) \in \{0, 1\}$

$\{j \in Readers : \ldots\} \in (\text{SUBSET } Readers)$

$$\text{THEOREM } SpecP \Rightarrow \Box[\forall i \in Readers : BeginRdP(i) \Rightarrow (\text{IF } p'[i] = 1 \text{ THEN } 1 \text{ ELSE } 0) \in \{0, 1\}]_{varsP}$$

$$\begin{aligned}\text{THEOREM } SpecP \Rightarrow \Box[\forall i \in Writers, cmd \in RegVals : \\ & DoWrP(i) \Rightarrow \\ & \quad \{j \in Readers : (rstate[j] \neq \langle\rangle) \\ & \qquad\qquad \land \ (p[j] = Len(rstate'[j]))\} \\ & \qquad \in (\text{SUBSET } Readers)]_{varsP}\end{aligned}$$

Temporarily end the module here to have *TLC* test the correctness of the preceding two theorems.

We now declare the variable s and define the initial predicate *InitPS* and next-state action *NextPS* of *SpecPS*. When stuttering steps are not being taken, s equals *top* , a value defined in module *Stuttering*. When stuttering steps are being taken, s equals a record with components:

$s.val$: Described above.

$s.id$: A value that identifies the action to which stutterin steps are being added.

$s.ctxt$: The value of bound identifiers in the context of the action for which the action is being executed.

VARIABLE s

$$varsPS \triangleq \langle vars, p, s\rangle$$

$$\text{INSTANCE } Stuttering \text{ WITH } vars \leftarrow varsP$$

$$
\begin{aligned}
TypeOKPS \triangleq\ & TypeOKP \land (s \in \{top\} \cup \\
& \quad [id : \{\text{"DoWr"}\}, \\
& \quad\ \ ctxt : Writers, \\
& \quad\ \ val : \text{SUBSET } Readers] \cup \\
& \quad [id : \{\text{"BeginRd"}\}, \\
& \quad\ \ ctxt : Readers, \\
& \quad\ \ val : \{0, 1\}])
\end{aligned}
$$

$$InitPS \triangleq InitP \land (s = top)$$

The actions *As* for each action *A* are defined using operators defined in the *Stuttering* module. The assumptions assert correctness conditions (2) and (3), described in a comment above, for the two actions to which stuttering steps are added.

$$
\begin{aligned}
BeginRdPS(i) \triangleq\ & MayPostStutter(BeginRdP(i), \text{"BeginRd"}, i, 0, \\
& \quad \text{IF } p'[i] = 1 \text{ THEN } 1 \text{ ELSE } 0, \\
& \quad \text{LAMBDA } j : j - 1)
\end{aligned}
$$

$$\text{ASSUME } StutterConstantCondition(\{0, 1\}, 0, \text{LAMBDA } j : j - 1)$$

$$BeginWrPS(i, cmd) \triangleq NoStutter(BeginWrP(i, cmd))$$

$$
\begin{aligned}
DoWrPS(i) \triangleq\ & MayPostStutter(DoWrP(i), \text{"DoWr"}, i, \{\}, \\
& \quad \{j \in Readers : \\
& \quad\quad (rstate[j] \neq \langle\rangle) \land (p[j] = Len(rstate'[j]))\}, \\
& \quad \text{LAMBDA } S : S \setminus \{\text{CHOOSE } x \in S : \text{TRUE}\})
\end{aligned}
$$

$$
\begin{aligned}
\text{ASSUME } & StutterConstantCondition(\text{SUBSET } Readers, \{\}, \\
& \quad \text{LAMBDA } S : S \setminus \{\text{CHOOSE } x \in S : \text{TRUE}\})
\end{aligned}
$$

$$IEndRdPS(i, j) \triangleq NoStutter(IEndRdP(i, j))$$

$$EndWrPS(i) \triangleq NoStutter(EndWrP(i))$$

$$
\begin{aligned}
NextPS \triangleq\ & \lor \exists i \in Readers : \lor BeginRdPS(i) \\
& \qquad\qquad\qquad\quad \lor \exists j \in 1 \mathrel{..} Len(rstate[i]) : IEndRdPS(i, j) \\
& \lor \exists i \in Writers : \lor \exists cmd \in RegVals : BeginWrPS(i, cmd) \\
& \qquad\qquad\qquad\quad \lor DoWrPS(i) \lor EndWrPS(i)
\end{aligned}
$$

For convenience, we give a name to the safety part of the specification as well as to the spec with its liveness condition.

$$
\begin{aligned}
& SafeSpecPS \triangleq InitPS \land \Box[NextPS]_{varsPS} \\
& SpecPS \triangleq SafeSpecPS \land Fairness
\end{aligned}
$$

We now define the refinement mapping. The externally visible variable *interface* is of course instantiated with the variable of the same name, as is the internal variable *mem*. The internal variable *istate* is instantiated by *istateBar*, defined here. The value of *istateBar*, like the value of *istate* in spec *Spec* of module *LinearSnapshot*, is a function with domain *Procs*, which we have defined in this moduule to equal $Readers \cup Writers$. For $i \in Writers$, we let $istateBar[i]$ equal $wstate[i]$. The value of $istate[i]$ for $i \in Readers$ is more complicated. It has to be defined so that the stuttering steps we added become the appropriate $DoRd(i)$ steps under the refinement mapping. You should study the definition to understand why they they are.

$$
\begin{array}{l}
istateBar \triangleq [i \in Readers \cup Writers \mapsto \\
\quad \textsc{if}\ i \in Writers \\
\quad\quad \textsc{then}\ wstate[i] \\
\quad\quad \textsc{else}\ \ \textsc{if}\ rstate[i] = \langle\rangle \\
\quad\quad\quad\quad \textsc{then}\ interface[i] \\
\quad\quad\quad\quad \textsc{else}\ \ \textsc{if}\ p[i] = 1 \\
\quad\quad\quad\quad\quad\quad \textsc{then}\ \textsc{if}\ \land s \neq top \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad \land s.id = \text{“BeginRd”} \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad \land s.ctxt = i \\
\quad\quad\quad\quad\quad\quad\quad\quad \textsc{then}\ NotMemVal \\
\quad\quad\quad\quad\quad\quad\quad\quad \textsc{else}\ \ rstate[i][1] \\
\quad\quad\quad\quad\quad\quad \textsc{else}\ \ \textsc{if}\ \lor p[i] > Len(rstate[i]) \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad \lor \land s \neq top \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad \land s.id = \text{“DoWr”} \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad \land i \in s.val \\
\quad\quad\quad\quad\quad\quad\quad\quad \textsc{then}\ NotMemVal \\
\quad\quad\quad\quad\quad\quad\quad\quad \textsc{else}\ \ rstate[i][p[i]]]
\end{array}
$$

The following INSTANCE statement and theorems assert that *SafeSpecPS* and *SpecPS* implement formulas *SafeSpec* and *Spec*, respectively, of module *LinearSnapshot*, under the refinement mapping.

$LS \triangleq \textsc{instance}\ LinearSnapshot\ \textsc{with}\ istate \leftarrow istateBar$

$\textsc{theorem}\ SafeSpecPS \Rightarrow LS!SafeSpec$

$\textsc{theorem}\ SpecPS \Rightarrow LS!Spec$

* Modification History
* Last modified Sat *Oct* 22 01:50:44 *PDT* 2016 by *lamport*
* Created *Wed Oct* 05 02:26:43 *PDT* 2016 by *lamport*