MODULE *Stuttering*

This module is explained in Section 5 of the paper "Auxiliary Variables in TLA+". It defines operators used to add a stuttering variable s to a specification *Spec* to form a specification *SpecS*. It is mean to be instantiated with s replaced by the stuttering variable to be added and *vars* replaced by the tuple of all variables in the original specification.

If *Init* is the initial predicate of *Spec* , then the initial predicate of *SpecS* is $Init \land (s = top)$, where top is defined below.

The next-state action of *SpecS* is obtained by replacing each subaction A of a disjunctive representation of the next-state action Next of *Spec* with an action AS written in terms of operators defined below. (Disjunctive representations are described in Section 3.2 of "Auxiliary Variables in TLA+".) Action AS executes A and stuttering steps added either before or after an A step. The basic idea is that s equals top except while stuttering steps are being taken, when it is a record with the following fields:

- *id*: A value that identifies the action A .
- *ctxt*: A value identifying the context under which A is executed. For example, if A appears in a formula $\exists\, i, j \in S : A$, this would equal the value of the pair $\langle i, j \rangle$ for which A is being executed.
- *val*: A value that is decremented by each stuttering step, until it reaches a minimum value.

The arguments of the operators defined in this module have the following meanings.

A
The subaction A of Next.

id
A string identifying action A .

Sigma
A set of values ordered by some "less-than" relation. This is the set of possible values of $s.val$ when executing stuttering steps before or after subaction A .

bot
The minimum element of *Sigma* under its less-than relation.

initVal
The initial value to which $s.val$ is set for executing stuttering steps before or after A .

decr
An operator such that each stuttering step changes $s.val$ to $decr(s.val)$.

context
The context in which A appears. It is the expression that is evaluated to determine the value to which $s.ctxt$ is set.

enabled
A formula equivalent to ENABLED A . You can always take it to be
ENABLED A , but you can usually find an expression that equals
ENABLED A in every reachable state of *Spec* but is easier for *TLC* to compute.

EXTENDS *Naturals*, *TLC*

$top \triangleq [top \mapsto \text{"top"}]$

VARIABLES s, *vars*

Equals AS when no stuttering steps are being added before or after A .

$$NoStutter(A) \triangleq (s = top) \land A \land (s' = s)$$

The $PostStutter$ and $PreStutter$ operators define actions that perform stuttering steps after executing A and before executing A , respectively. If $bot = 1$, $initVal = 42$, and $decr(i) = i - 1$, then the actions they define add 42 stuttering steps. They always add at least one stuttering step.

$$\begin{array}{l}
PostStutter(A,\ actionId,\ context,\ bot,\ initVal,\ decr(_)) \triangleq \\
\quad \textsc{if}\ s = top\ \textsc{then}\ \begin{array}[t]{l} \land A \\ \land s' = [id \mapsto actionId,\ ctxt \mapsto context,\ val \mapsto initVal] \end{array} \\
\qquad\qquad\qquad\ \ \textsc{else}\ \begin{array}[t]{l} \land s.id = actionId \\ \land \textsc{unchanged}\ vars \\ \land s' = \textsc{if}\ s.val = bot\ \textsc{then}\ top \\ \qquad\qquad\qquad\qquad\quad \textsc{else}\ [s\ \textsc{except}\ !.val = decr(s.val)] \end{array}
\end{array}$$

$$\begin{array}{l}
PreStutter(A,\ enabled,\ actionId,\ context,\ bot,\ initVal,\ decr(_)) \triangleq \\
\quad \textsc{if}\ s = top \\
\qquad \textsc{then}\ \begin{array}[t]{l} \land enabled \\ \land \textsc{unchanged}\ vars \\ \land s' = [id \mapsto actionId,\ ctxt \mapsto context,\ val \mapsto initVal] \end{array} \\
\qquad \textsc{else}\ \begin{array}[t]{l} \land s.id = actionId \\ \land \textsc{if}\ s.val = bot\ \textsc{then}\ \begin{array}[t]{l} \land s.ctxt = context \\ \land A \\ \land s' = top \end{array} \\ \qquad\qquad\qquad\qquad\quad \textsc{else}\ \begin{array}[t]{l} \land \textsc{unchanged}\ vars \\ \land s' = [s\ \textsc{except}\ !.val = decr(s.val)] \end{array} \end{array}
\end{array}$$

The operators $MayPostStutter$ and $MayPreStutter$ are like $PostStutter$ and $PreStutter$, except they add one fewer stuttering step. For example, if $bot = 1$, $initVal = 42$, and $decr(i) = i - 1$, then they add 42 steps. If $initVal = bot$, then they simply execute A without any added stuttering steps.

$$\begin{array}{l}
MayPostStutter(A,\ actionId,\ context,\ bot,\ initVal,\ decr(_)) \triangleq \\
\quad \textsc{if}\ s = top\ \textsc{then}\ \begin{array}[t]{l} \land A \\ \land s' = \textsc{if}\ initVal = bot \\ \qquad\qquad\ \textsc{then}\ s \\ \qquad\qquad\ \textsc{else}\ [id \mapsto actionId,\ ctxt \mapsto context, \\ \qquad\qquad\qquad\quad val \mapsto initVal] \end{array} \\
\qquad\qquad\qquad\ \ \textsc{else}\ \begin{array}[t]{l} \land s.id = actionId \\ \land \textsc{unchanged}\ vars \\ \land s' = \textsc{if}\ decr(s.val) = bot \\ \qquad\qquad\ \textsc{then}\ top \\ \qquad\qquad\ \textsc{else}\ [s\ \textsc{except}\ !.val = decr(s.val)] \end{array}
\end{array}$$

$$\begin{array}{l}
MayPreStutter(A,\ enabled,\ actionId,\ context,\ bot,\ initVal,\ decr(_)) \triangleq \\
\quad \textsc{if}\ s = top \\
\qquad \textsc{then}\ \begin{array}[t]{l} \land enabled \\ \land \textsc{if}\ initVal = bot \end{array}
\end{array}$$

$$
\begin{array}{l}
\quad\quad\quad\quad \textsc{then}\ A \land (s' = s) \\
\quad\quad\quad\quad \textsc{else}\ \land s' = [id \mapsto actionId,\ ctxt \mapsto context,\ val \mapsto initVal] \\
\quad\quad\quad\quad\quad\quad\ \land \textsc{unchanged}\ vars \\
\textsc{else}\ \land s.id = actionId \\
\quad\quad\ \land \textsc{if}\ s.val = bot\ \textsc{then}\ \land s.ctxt = context \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\ \land A \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\ \land s' = top \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad\ \textsc{else}\ \land \textsc{unchanged}\ vars \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\ \land s' = [s\ \textsc{except}\ !.val = decr(s.val)]
\end{array}
$$

The following operator is true iff repeatedly apply *decr* to any element *sig* of *Sigma* produces a sequence *sig*, *decr*(*sig*), *decr*(*decr*(*sig*)), ... that eventually arrives at *bot*. This condition should be satisfied by the values of *Sigma*, *bot*, and *decr* used with the operators defined above to add stuttering steps to an action.

$$
\begin{array}{l}
StutterConstantCondition(Sigma,\ bot,\ decr(\_)) \triangleq \\
\quad \textsc{let}\ InverseDecr(S) \triangleq \{sig \in Sigma \setminus S : decr(sig) \in S\} \\
\quad\quad\quad R[n \in Nat] \triangleq \textsc{if}\ n = 0\ \textsc{then}\ \{bot\} \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad \textsc{else}\ \textsc{let}\ T \triangleq R[n-1] \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad \textsc{in}\ \ T \cup InverseDecr(T) \\
\quad \textsc{in}\ \ Sigma = \textsc{union}\ \{R[n] : n \in Nat\}
\end{array}
$$

TLC can evaluate *StutterConstantCondition* only in a model that replaces *Nat* by the set $0 \,..\, n$ for some integer n. The following operator is equivalent to *StutterConstantCondition* when *Sigma* is a finite set, but doesn't require modifying any definitions.

$$
\begin{array}{l}
AltStutterConstantCondition(Sigma,\ bot,\ decr(\_)) \triangleq \\
\quad \textsc{let}\ InverseDecr(S) \triangleq \{sig \in Sigma \setminus S : decr(sig) \in S\} \\
\quad\quad\quad ReachBot[S \in \textsc{subset}\ Sigma] \triangleq \\
\quad\quad\quad\quad \textsc{if}\ InverseDecr(S) = \{\}\ \textsc{then}\ S \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad \textsc{else}\ ReachBot[S \cup InverseDecr(S)] \\
\quad \textsc{in}\ \ ReachBot[\{bot\}] = Sigma
\end{array}
$$

---

* Modification History
* Last modified Sat *Dec* 31 17:47:02 *PST* 2016 by *lamport*
* Created *Tue* *Dec* 08 11:51:34 *PST* 2015 by *lamport*