The definition of sequential \texttt{FUN} in Figure 6 needs to change as follows:
1) add \texttt{callcc} and its two rules below
2) replace structural operators with the ones in the picture to the right
3) replace eval and result definitions as below
4) declare non-deterministic the rules for write, variable lookup, read and print
5) add six more rules for threads:
a) one rule for creation of threads
   and one for termination of threads;
b) two rules for acquiring a lock
   c) two rules for releasing a lock
No other changes needed.

\begin{figure}[h]
\centering
\includegraphics[width=0.8\textwidth]{fig7.png}
\caption{Adding call/cc and threads to FUN}
\end{figure}

as to throwing exceptions, exactly like in the original context. Therefore, in our language, \texttt{call/cc}
appears to better abbreviate “call with current context”.

Even though \texttt{call/cc} is conceptually more powerful than our other control-intensive language
constructs (their translation would be technically involved, though), it is actually very easy to
define in our framework. A special “current context” value is needed, so we define an operation
\texttt{cc : Continuation \times Ctrl \times Environment \rightarrow Val}, as well as a corresponding continuation item,
\texttt{callcc : \rightarrow ContinuationItem}. Note that \texttt{callcc} is declared strict in its argument. When a value
(expected to be a function closure, namely the evaluated argument expression of \texttt{callcc}) is passed
to it at the top of the continuation, a special “control context” value is created and passed to its
argument function:

\[
(k( V : Val \Rightarrow callcc \Rightarrow K ) \Rightarrow Ctrl ) \Rightarrow Env
\]

If the special control context value is ever passed a value, then the original execution context is