Diller’s Text Editor Specification (Chap. 18)

This example develops a simple display-oriented text-editor. The “text” is a linear stream of characters, some of which are control characters that induce the familiar two-dimensional line structure on the data characters. A similar (though not identical) system was the object of algebraic specification in our mid-term exam.

This specification views a “text” as a sequence of characters (including the “newline” character ↓), plus a “cursor” placed somewhere within this sequence. Operations are understood to operate relative to the current cursor position, and change the position of the cursor as well as the text. The structure is modeled as a pair of character sequences, the subsequences to the left and right of the cursor. Diller develops the specification in several levels — we’ll examine the 'Doc2' version.

The first components of the specification are a basic type for the (abstract) character set

\[
[\text{Char}]
\]

and the state space schema

\[
\text{Doc2} \quad \text{Doc2}
\]

\[
\begin{align*}
\text{left, right: seq Char} \\
\downarrow: \text{Char} \\
\text{Line == seq(Char } \setminus \{\downarrow\}\text{)} \\
\text{doclines: seq Line} \\
\text{line, col: N}_1
\end{align*}
\]

\[
\begin{align*}
\text{line} & \leq \#\text{doclines} \\
\text{col} & \leq \#(\text{doclines line}) + 1 \\
\text{left} \wedge \text{right} & = \text{flatten doclines} \\
\text{line} & = \#(\text{left } \setminus \{\downarrow\}) + 1 \\
\text{col} & = \#\text{left} - \#(\text{flatten(doclines for (line-1))})
\end{align*}
\]
Here for and flatten are the (global) functions defined by

\[ \text{flatten: seq}_1 \text{ Line } \rightarrow \text{ seq Char} \]
\[ \_ \text{ for } _\_ : \text{ seq X } \times \text{ N } \rightarrow \text{ seq X} \]

\[ \forall \text{xs: Line; xss: seq}_1 \text{ Line} \cdot \]
\[ \quad \text{flatten } <\text{xs}> = \text{xs} \land \]
\[ \quad \text{flatten}(<\text{xs}> ^ \text{xss}) = \text{xs} ^ <\downarrow> ^ (\text{flatten xss}) \]
\[ \text{xs for n = 1..n < xs} \]

Now we can examine schemas for some selected operations.

\[ \text{DeleteLeftDoc2} \]
\[ \Delta \text{Doc2} \]

left ≠ < >
left' = front left
right' = right

The specification of (the normal case of) this operation is simple because the state space invariant assures all the other post-state variable values are uniquely determined — the expansion showing all the post-state variable values is shown in Diller. Diller also adds a specification for the exceptional case.

\[ \text{ErrorAtTop} \]
\[ \Xi \text{Doc2} \]
\[ \text{rep!}: \text{ Report} \]

left = < >
rep! = 'At top of document'

Then the complete operation schema for the delete-left operation is

\[ \text{DoDeleteLeftDoc2} \triangleq \text{DeleteLeftDoc2} \land \text{Success} \lor \text{ErrorAtTop}. \]
Other operations are also easily specified. For instance, the operation to move the
cursor one character to the left.

\[\text{MoveLeftDoc2} \]
\[\Delta \text{Doc2} \]
\[
\begin{align*}
\text{left} & \neq \langle > \\
\text{left}' & = \text{front left} \\
\text{right}' & = \langle \text{last left} > ^\wedge \text{right}
\end{align*}
\]

Again, the (normal case) specification of this operation is simple because the
state space invariant assures all the other post-state variable values are uniquely
determined. The exceptional case is treated similarly to the previous case.

The other operations for inserting or deleting characters from the left or right, and
moving the cursor right one character are also straightforward. The operations for
moving up or down one line require some consideration. Unfortunately, Diller
does not complete the specification and omits these operations.

We will explore the specification of moving the cursor up one line. This
development is also eased by the elaborate state invariant. We can describe the
post-state for a subset of state variables from which all the other post-state
values can be deduced.

\[\text{CursorUpDoc2} \]
\[\Delta \text{Doc2} \]
\[
\begin{align*}
\text{line} & > 1 \\
\text{doclines}' & = \text{doclines} \\
\text{line}' & = \text{line} - 1 \\
\text{col}' & = \min\{\text{col}, \#(\text{doclines line}') + 1\}
\end{align*}
\]

While the cursor may move, the document content (doclines) is unchanged. The
post-state line and column numbers are determined by the given post-conditions.
From the invariant, \#\text{left}' = \text{col}' + \#(\text{flatten(doclines'} for (\text{line'} - 1))), and therefore
\#\text{left}' and hence \text{left}' are known. But then the invariant condition \text{left}' ^\wedge \text{right}' =
\text{doclines'} determines \text{right}', and all the post-state values are determined.
The possibility of not providing complete definitions of the post-state for all the variables in the state space can substantially ease the creation of \( \mathbf{Z} \) specifications. A specification will not be complete, if some of these values remain indeterminate, but if variables not mentioned explicitly can have their post-state values inferred from the state invariant and the values of the subset of variables that are defined, a specification still meets the goals we set. Since the variables whose post-state values are most intuitive to describe may vary with the operation, this is a useful feature to keep in mind. It shows itself to be quite helpful here — especially compare the specification of this last operation with the corresponding operation in the algebraic specification in our midterm exam (see our class directory).