Package `sml`

Allen Leung

May 17, 2018

1 Introduction

The `sml` package defines a `verbatim`-like environment called `smldisplay` for typesetting Standard ML programs. Like the `alltt` environment, backslashes '\' and the braces { and } have their usual meaning in `smldisplay`, so it is possible to use other macros and commands within the `smldisplay` environment. Meta-characters such as #, %, $, _ and ^ are disabled and appears verbatim.

To enter math mode, the user can use `\(` ...`)` or `\[` ...`\]`. But unlike the `alltt` environment, the superscripts ^ and subscripts _ characters are available inside math mode.

The character `'` is interpreted as the beginning of a ML type variable. Type variables are typeset in italics within the `smldisplay` environment. For example,

```
\begin{smldisplay}
   datatype 'a tree = EMPTY
                    | NODE of 'a * 'a tree list
\end{smldisplay}
```

is typeset as follows:

```
   datatype 'a tree = EMPTY
                    | NODE of 'a * 'a tree list
```

The environment `smlboxeddisplay` is similar to `smldisplay` except that a boxed is also drawn around the displayed program. For example, if we write:

```
\begin{smlboxeddisplay}
   datatype 'a tree = EMPTY
                    | NODE of 'a * 'a tree list
\end{smlboxeddisplay}
```

we get:

```
   datatype 'a tree = EMPTY
                    | NODE of 'a * 'a tree list
```

1.1 Highlighting keywords

A similar environment, called `smldisp`, can be used to highlight all SML keywords. However, math mode and other macros are *unavailable* in this environment. For example, in `smldisp` we can write:

```
\begin{smldisp}
   (* A n-ary tree *)
   datatype 'a tree = EMPTY
                    | NODE of 'a * 'a tree list
   (* Flatten a tree as a list in preorder *)
   fun flatten(EMPTY) = []
     | flatten(NODE(x,children)) = [x] @ List.concat(map flatten children)
\end{smldisp}
```

and get the following result:

```
   (* A n-ary tree *)
   datatype 'a tree = EMPTY
                    | NODE of 'a * 'a tree list
   (* Flatten a tree as a list in preorder *)
   fun flatten(EMPTY) = []
     | flatten(NODE(x,children)) = [x] @ List.concat(map flatten children)
```

Note that the keywords "datatype" and "of" have been typeset as **datatype** and **of**. Furthermore, comments are typeset in small italics font.

The following macros control how keywords and comments are typeset in this environment:

```
\newcommand{\makeSmlKeyword}[1]{{\bf #1}}
\newcommand{\smlCommentSize}{\small}
\newcommand{\smlCommentFont}{\it}
\newcommand{\BeginSmlComment}{\begingroup\smlCommentSize\smlCommentFont}
\newcommand{\EndSmlComment}{\endgroup}
```

These can be redefined by the user if necessary.

1.2 Type Variable Translations

It is possible to define type variable translations for `smldisplay` and `smldisp` environments. For example, if we write:

```
   \smlTypeVar{a}{\(\alpha\)}
   \smlTypeVar{foo}{\(\underline\beta\)}
   \begin{smldisplay}
      datatype 'a tree = EMPTY | NODE of 'a * 'a tree list
      type 'foo foo = ('foo * 'foo) tree
      type 'c seq = 'c list
   \end{smldisplay}
```

we get:

datatype α'*a* tree = EMPTY | NODE of α'*a* * α'*a* tree list
type $\underline{\beta}$'*foo* foo = ($\underline{\beta}$'*foo* * $\underline{\beta}$'*foo*) tree
type '*c* seq = '*c* list

Note that all occurrances of `'a` has been translated into α, while all occurrances of `'foo` has been translated into $\underline{\beta}$.

A type variable translation declared by `smlTypeVar` is active in its scope until it is removed by the macro `\smlRemoveTypeVar`. For example, we can write:

```
\smlRemoveTypeVar{foo}
```

to remove the translation on type variable `'foo`.

1.3 `\verb`-like macros

A `\verb`-like macro called `\sml` is available for typesetting short SML program fragments within running text. For example, we can write the following:

```
\begin{quotation}
   The datatype \sml{'a tree} implements a polymorphic n-ary tree.
The function \sml{val rev : 'a tree -> 'a list} flattens a tree into a list.
\end{quotation}
```

and obtain:

> The datatype α'*a* `tree` implements a polymorphic n-ary tree. The function `val rev :` α'*a* `tree ->` α'*a* `list` flattens a tree into a list.

The macro `\sml` behaves very much like the `smldisplay` environment, except that newlines are not interpreted verbatim.

Similarly, there is a `\verb`-like macro called `\Sml` that behaves like the `smldisp` environment. For example, writing

```
\begin{quotation}
   The datatype \Sml{'a tree} implements a polymorphic n-ary tree.
The function \Sml{val rev : 'a tree -> 'a list} flattens a tree into a list.
\end{quotation}
```

we obtain:

> The datatype α'*a* `tree` implements a polymorphic n-ary tree. The function **val** `rev :` α'*a* `tree ->` α'*a* `list` flattens a tree into a list.

1.4 Changing the Fonts

The macros `\smlFont` and `\smlTypeVarFont` define the fonts used for typesetting ML text and type variables. They are predefined as follows:

```
   \newcommand{\smlFont}{\verbatim@font}
   \newcommand{\smlTypeVarFont}{\it}
```

Furthermore, the default method of typesetting a type variable is defined as:

```
   \newcommand{\makeSmlTypeVar}[1]{'{\smlTypeVarFont #1}}
```

These can be overridden by the user if desired.

1.5 Enabling $

By default, the math shift character `$` is disabled within the environment `smldisplay` and the macro `sml`. It is possible to enable this character by declaring:

```
   \smlDollarOn
```

in the prologue of a document. For example, we can write:

```
\smlDollarOn
\begin{smldisplay}
   datatype 'a tree = EMPTY | NODE of 'a * 'a tree list
   \textrm{A balanced tree with $n$ nodes has height $O(\log n)$}
\end{smldisplay}
```

and obtain:

```
   datatype α'a tree = EMPTY | NODE of α'a * α'a tree list
```
A balanced tree with n nodes has height $O(\log\ n)$

To turn off the math shift character `$`, we can write

```
   \smlDollarOff
```

1.6 Numbered Program Listings

Numbered program listings can be displayed using the `smllisting` environment, which behaves exactly like `smldisplay` except that every line is prefixed by a line number. For example, when we write:

```
\smlTypeVar{n}{\(\alpha\)}
\smlTypeVar{e}{\(\beta\)}
\smlTypeVar{g}{\(\gamma\)}
\begin{smllisting}{1}{1}
signature SINGLE_SOURCE_SHORTEST_PATHS =
sig
```

```
   val single_source_shortest_paths :
                 \{ weight : 'e Graph.edge -> 'w,
                   <      : 'w * 'w -> bool,
                   +      : 'w * 'w -> 'w,
                   zero   : 'w,
                   inf    : 'w
                 \} ->
                 ('n,'e,'g) Graph.graph ->
                 Graph.node_id ->
                 \{ dist : 'w Array.array,
                   pred :  Graph.node_id Array.array
                 \}
end
\end{smllisting}
```

we get:

```
signature SINGLE_SOURCE_SHORTEST_PATHS =
sig

   val single_source_shortest_paths :
                 { weight : β'e Graph.edge -> 'w,
                   <      : 'w * 'w -> bool,
                   +      : 'w * 'w -> 'w,
                   zero   : 'w,
                   inf    : 'w
                 } ->
                 (α'n,β'e,γ'g) Graph.graph ->
                 Graph.node_id ->
                 { dist : 'w Array.array,
                   pred :  Graph.node_id Array.array
                 }
end
```

The environment `smllisting` requires two numeric parameters. The first parameter determines the initial line number of the listing, while the second parameter determines how often the line number should be printed. For example, if the second parameter is 2, then the line number appears every two lines.

The environment `smlboxedlisting` is similar to `smllisting` except that a box is also drawn around the program listing.

The following macros control how the numbers are displayed

```
\newcommand{\smlNumberFont}{\smlFont}
\newcommand{\smlNumberStyle}[1]{\arabic{#1}}
```

The first macro `\smlNumberFont` controls the font used for line numbering, which by default is `\tt`. The second macro `\smlNumberStyle` displays the line count as arabic numerals.