fun {FoldL Xs F S}
  case Xs of nil then
    S
  | X|Xr then
    {FoldL Xr F {F S X}}
  end
end

fun {FoldR Xs F S}
  case Xs of nil then
    S
  | X|Xr then
    {F X {FoldR Xr F S}}
  end
end
The idea is to show how \textbf{FoldL} and \textbf{FoldR} will execute given the following function definitions:

\begin{verbatim}
fun \{Cons X Xr\} X|Xr end
fun \{Snoc Xr X\} X|Xr end
\end{verbatim}

The following examples will be demonstrated:

\{Browse \{FoldL [a b c d] Snoc nil\}\}
\{Browse \{FoldR [a b c d] Cons nil\}\}
\{Browse \{FoldL [a b c d] Cons nil\}\}
\{Browse \{FoldR [a b c d] Snoc nil\}\}

\{FoldL [a b c d] Snoc nil\}
\{Snoc nil a\}
a|nil
\{Snoc a|nil b\}
b|a|nil
\{Snoc b|a|nil c\}
c|b|a|nil
\{Snoc c|b|a|nil d\}
d|c|b|a|nil
\[d c b a\]


Our first call to the function \textbf{FoldL} with the function \textbf{Snoc} (see definition above) and initial accumulator initiated to \textit{nil}. It will recurse the list \([a b c d]\) \textbf{from left to right}:

\textbf{FoldL} will call our function \textbf{Snoc} with the accumulator \((\textit{nil})\) and the head of the list \((a)\)
\textbf{Snoc} will simply return \(X|Xr\), which yields to the new accumulator value \(a|\textit{nil}\)
\textbf{FoldL} will now recurse and call our function \textbf{Snoc} with the accumulator \((a|\textit{nil})\) and the head of the current list \((b)\)
\textbf{Snoc} will simply return \(X|Xr\), which yields to the new accumulator value \(b|a|\textit{nil}\)
\textbf{FoldL} will now recurse and call our function \textbf{Snoc} with the accumulator \((b|a|\textit{nil})\) and the head of the current list \((c)\)
\textbf{Snoc} will simply return \(X|Xr\), which yields to the new accumulator value \(c|b|a|\textit{nil}\)
\textbf{FoldL} will now recurse and call our function \textbf{Snoc} with the accumulator \((c|b|a|\textit{nil})\) and the head of the current list \((d)\)
\textbf{Snoc} will simply return \(X|Xr\), which yields to the new accumulator value \(d|c|b|a|\textit{nil}\)

Since \textbf{FoldL} has recursed the whole list \([a b c d]\), it is now done and returns the final accumulator, and we have our result!
Our first call to the function \texttt{FoldR} with the function \texttt{Cons} (see definition above) and initial accumulator initiated to \texttt{nil}. It will recurse the list \{a b c d\} \textbf{from right to left}.

\texttt{FoldR} will call our function \texttt{Cons} with the accumulator (\texttt{nil}) and end of the list (d) (notice the order is different for \texttt{FoldR}).

\texttt{Cons} will simply return \texttt{X|Xr}, which yields to the new accumulator value \texttt{d|nil}.

\texttt{FoldR} will now recurse and call our function \texttt{Cons} with the accumulator (\texttt{d|nil}) and the end of the current list (c).

\texttt{Cons} will simply return \texttt{X|Xr}, which yields to the new accumulator value \texttt{c|d|nil}.

\texttt{FoldR} will now recurse and call our function \texttt{Cons} with the accumulator (\texttt{c|d|nil}) and the end of the current list (b).

\texttt{Cons} will simply return \texttt{X|Xr}, which yields to the new accumulator value \texttt{b|c|d|nil}.

\texttt{FoldL} will now recurse and call our function \texttt{Cons} with the accumulator (\texttt{b|c|d|nil}) and the end of the current list (a).

\texttt{Cons} will simply return \texttt{X|Xr}, which yields to the new accumulator value \texttt{d|c|b|a|nil}.

Since \texttt{FoldR} has recursed the whole list \{a b c d\}, it is now done and returns the final accumulator, and we have our result!
Notice that returning $X|Xr$ is the same as returning the tuple $'|'(X|Xr)$. So if $X='|'(Y|Yr)$, then $'|'(X|Xr)$ yields to $'|'|'(Y|Yr)|Xr$ which is the same as $(Y|Yr)|Xr$

Our first call to the function `FoldL` with the function `Cons` (see definition above) and initial accumulator initiated to `nil`. It will recurse the list [a b c d] from left to right.

`FoldL` will call our function `Cons` with the accumulator `(nil)` and the head of the list `(a)`

`Cons` will simply return $X|Xr$. OBS! Cons has a different order for its parameters and therefore the value returned will be `nil|a`.

`FoldL` will now recurse and call our function `Cons` with the accumulator `(nil|a)` and the head of the current list `(b)`

`Cons` will simply return $X|Xr$, which yields to the new accumulator value `(nil|a)|b`.

`FoldL` will now recurse and call our function `Cons` with the accumulator `(nil|a)|b` and the head of the current list `(c)`

`Cons` will simply return $X|Xr$, which yields to the new accumulator value `((nil|a)|b)|c`.

`FoldL` will now recurse and call our function `Cons` with the accumulator `((nil|a)|b)|c` and the head of the current list `(d)`

`Cons` will simply return $X|Xr$, which yields to the new accumulator value `(((nil|a)|b)|c)|d`.

Since `FoldL` has recursed the whole list [a b c d], it is now done and returns the final accumulator, and we have our result!

ali@sics.se
Notice that returning $X|Yr$ is the same as returning the tuple
’( X Xr). So if $X=’(Y Yr)$, then ’(X Xr) yields to
’(‘(Y Yr) Xr) which is the same as (Y|Yr)|Xr

Our first call to the function $\text{FoldR}$ with the function $\text{Snoc}$
(se definition above) and initial accumulator initiated to nil.
It will recurse the list [a b c d] from right to left!

$\text{FoldR}$ will call our function $\text{Snoc}$ with the accumulator
(nil) and the end of the list (d)

$\text{Snoc}$ will simply return $X|Xr$. OBS! Cons has a
different order for its parameters and therefore the value
returned will be nil|d

$\text{FoldR}$ will now recurse and call our function $\text{Snoc}$ with
the accumulator (nil|d) and the end of the current list (c)

$\text{Snoc}$ will simply return $X|Xr$, which yields to the new
accumulator value (nil|d)|c

$\text{FoldR}$ will now recurse and call our function $\text{Snoc}$ with
the accumulator (nil|d)|c and the end of the current list (b)

$\text{Snoc}$ will simply return $X|Xr$, which yields to the new
accumulator value (((nil|d)|c)|b)

Since $\text{FoldR}$ has recursed the whole list [a b c d], it is
now done and returns the final accumulator, and we
have our result!

ali@sics.se