PBookParser.g

takashi

October 31, 2008

Contents

1 Introduction **1**

2 PBook Parser **1**

3 HTML Renderer **2**

4 TeX Renderer **3**

1 Introduction

PBookParser.g is a parser of pbook for FoNC written in jolt3 grammar. It is a subclass of Parser. This is called through a driver program pbook.st. Instance variables 'user' and 'fileName' are given by the driver program, and they are used for the title of the generated document.

```
PBookParser : Parser (user fileName)
```

The default entry point start defined for convenience.

```
start = tex
```

The parser consists two parts. A parser parses .st or .g source code and generates a parse tree, and a render build formatted text. This program includes two renders, HTML render and TeX render.

2 PBook Parser

PBook Parser builds a simple syntax tree from source file. The structure of syntax tree is ((type line line...) ...) where type is ether header1, header2, header3, comment, body, or emptyLine, and the contents are followed. The pbook parser is line oriented, so each selction is delimited by an empty line.

```
parse = ( emptyLine | comment | body )*

emptyLine = eol -> `(emptyLine)
```

There are two kinds of comment. A block comment is encompassed by double quotation mark, it is used in .st source. And a line comment is a line start with %, it is used .g source. This parser supports Both styles.

```
comment = (blockComment | lineComment):lines <findCommentLevel lines>

findCommentLevel = #(#(commentLevel:x) .*:xs) -> `(,@x ,@xs)
```

```
commentLevel = '###' .*:chars -> `(header3 ,chars)
    | '##' .*:chars -> `(header2 ,chars)
    | '#' .*:chars -> `(header1 ,chars)
    | .*:chars -> `(comment ,chars)

blockComment =  '"' (!'"' .)+:x '"' (!eol .)* eol? -> `(,x)
lineComment = ('%' (!eol .)*:c eol? :ret -> `(,@c ,@ret))+
```

A section except comment is regarded as a body, and rendered in a fix witdth font.

```
body = line+:lines -> `(body ,@lines)
line = !comment (!eol .)* :chars eol?:ret -> `(,@chars ,@ret)

eol   = '\r\n' | '\n' | '\r'
eof   = !.
```

3 HTML Renderer

renderHtml renderes HTML formetted text. header is rendered as <h1> .. <h3> element.

```
html = <renderHtml <parse>>

renderHtml = htmlHeader #(#(renderHtmlSection)*)

htmlHeader = { '<style type="text/css"> pre { background-color: #ECF0F0 } </style>' put }

renderHtmlSection = #header1 .*:lines <renderHtmlTag 'h1' lines>
    | #header2 .*:lines <renderHtmlTag 'h2' lines>
    | #header3 .*:lines <renderHtmlTag 'h3' lines>
    | #comment .*:lines <renderHtmlTag 'p' lines>
    | #body .*:lines <renderHtmlTag 'pre' lines>
    | #emptyLine .* { '\n' put }
```

renderHtmlTag renderes HTML element. The first argument is a tag name and second argument is a list of lines.

```
renderHtmlTag = .:tag .:lines
        { ('<', tag, '>') putln. }
        <renderHtmlLines lines>
        { ('</', tag, '>') putln. }

renderHtmlLines = #(#(renderHtmlChar*)*)
```

renderHtmlChar defines conversion rule for HTML escape.

```
renderHtmlChar = '<' { '&lt;' put }
    | '>' { '&gt;' put }
    | '"' { '&quot;' put }
    | '&' { '&amp;' put }
    | .:char { char put }
```

4 TeX Renderer

```
tex = <renderTex <parse>>

renderTex = texHeader #(#(renderTexSection)*) texFooter

texHeader = {
('
\\documentclass[notitlepage,a4paper]{article}
\\usepackage{fullpage}
\\title{', fileName, '}
\\author{', user, '}
\\begin{document}
\\maketitle
\\tableofcontents
') put }

texFooter = { '\\end{document}' putln }

renderTexSection = #header1 .*:lines <renderTexTag 'section' lines>
    | #header2 .*:lines <renderTexTag 'subsection' lines>
    | #header3 .*:lines <renderTexTag 'subsubsection' lines>
    | #comment .*:lines <renderTexLines lines>
    | #body .*:lines <renderTexVerbatim lines>
    | #emptyLine .* { '\n' put }
```

renderTexTag renderes TeX header element.

```
renderTexTag = .:tag .:lines
        { ('\n\\', tag, '\173') put }
        <renderTexLines lines>
        { '\175' putln }

renderTexVerbatim = .:lines
        { '\\begin\{verbatim\}' putln }
        <renderTexVerbatimLines lines>
        { '\\end\{verbatim\}' putln }

renderTexLines = #(#(renderTexChar*)*)
```

Normal tex escaping rules

```
renderTexChar = ('#'|'%'|'&'|'~'|'$'|'_'|'^'|'{'|'}'):char -> { '\\' put. char put }
    | ('<'|'>'):char -> { $$ put. char put. $$ put }
    | .:char { char put }
```

The escaping rules for verbatim environments are unclear to me, too. So I just ignore here.

```
renderTexVerbatimLines = #(#(renderTexVerbatimChar*)*)
renderTexVerbatimChar = .:char { char put }
```