$SPAD/src/algebra axserver.spad

Arthur C. Ralfs

February 26, 2018

Abstract

The AxiomServer package is designed to provide a web interface to axiom.

Contents

1 Lisp preliminaries

Extract the lisp to a file named, say, http.lisp with the command

```
notangle -RServer$\backslash$ Lisp axserver.pamphlet > http.lisp
```

⟨*Server Lisp*⟩≡
```
;; file: http.lisp

(defun |Open| (path)
 (si::open path :direction :input :if-exists nil :if-does-not-exist nil)
 )

(defvar |StandardOutput| *standard-output*)

(defvar  |NewLine| '#\NewLine)

(defun |WriteLine| (string &optional (outstream *standard-output*))
 (write-line string outstream)
 (finish-output outstream) )

;; some regexp stuff

(defun |StringMatch| (s1 s2)
 (si::string-match s1 s2)
 )

(defun |ListMatches| (&rest args)
 (si::list-matches args)
 )

(defun |MatchBeginning| (i)
 (si::match-beginning i)
 )

(defun |MatchEnd| (i)
 (si::match-end i)
 )

;; the socket stuff

(defun |SiSock| (p spadfn)
;;  (format t "SiSocket-1")
 (si::socket p :server
            (function
             (lambda (w) (SPADCALL w spadfn) )
             )
```

```
           :daemon nil)
 )

(defun |SiListen| (s)
;;  (format t "SiListen-1")
 (si::listen s)
 )
(defun |SiAccept| (s) (si::accept s))
(defun |SiCopyStream| (q s) (si::copy-stream q s))

;; Camm Maguire's modified demo server

(defun foo (s)
 (setq get "" pathvar "")
 (do ((c (read-char s) (read-char s)))
    ((eq c '#\Space))
    (setq get (concat get (string c)))
    )
 (write-line "get: ")
 (write-line get)
 (do ((c (read-char s) (read-char s nil 'the-end)))
    ((eq c '#\Space))
    (setq pathvar (concat pathvar (string c)))
    )
 (write-line "pathvar: ")
 (write-line pathvar)
 (when pathvar
  (if (pathname-name (pathname pathvar))
      (with-open-file (q pathvar) (si::copy-stream q s))
    (dolist (l (directory pathvar)) (format s "~a~%" (namestring l)))
    )
  )
 (close s)
 )

(defun bar (p fn)
 (let ((s (si::socket p :server fn)))
      (tagbody l
              (when (si::listen s)
                      (let ((w (si::accept s)))
                              (foo w)))
              (sleep 10000)
              (go l))))

;;(bar 8080 #'foo)
```

2 Axiom Server

Extract the AxiomServer package with the command

`notangle axserver.pamphlet > axserver.spad`

⟨*package AXSERV AxiomServer*⟩≡

```
 )abbrev package AXSERV AxiomServer
 AxiomServer: public == private where

  public == with

   axServer: (Integer, SExpression->Void) -> Void
   multiServ: SExpression -> Void

  private == add

   getFile: (SExpression,String) -> Void
   getCommand: (SExpression,String) -> Void
   lastStep: () -> String
   lastType: () -> String
   formatMessages: String -> String
   getContentType: String -> String


   axServer(port:Integer,serverfunc:SExpression->Void):Void ==
     WriteLine("socketServer")$Lisp
     s := SiSock(port,serverfunc)$Lisp
     -- To listen for just one connection and then close the socket
     -- uncomment i := 0.
     i:Integer := 1
     while (i > 0) repeat
       if not null?(SiListen(s)$Lisp)$SExpression then
         w := SiAccept(s)$Lisp
         serverfunc(w)
 --        i := 0

   multiServ(s:SExpression):Void ==
         WriteLine("multiServ begin")$Lisp
         headers:String := ""
         char:String
         -- read in the http headers
         while (char :=
         STRING(READ_-CHAR_-NO_-HANG(s,NIL$Lisp,'EOF)$Lisp)$Lisp) ^= "EOF" and
          char ^= "NIL" repeat
 --           WriteLine$Lisp "multiServ while"char
```

```
            headers := concat [headers,char]
          WriteLine$Lisp headers
          StringMatch("([^ ]*)", headers)$Lisp
          u:UniversalSegment(Integer)
          u := segment(MatchBeginning(1)$Lisp+1,MatchEnd(1)$Lisp)$UniversalSegment(Integ
          reqtype:String := headers.u
          WriteLine$Lisp  concat ["request type: ",reqtype]
          if  reqtype = "GET" then
              StringMatch("GET ([^ ]*)",headers)$Lisp
              u:UniversalSegment(Integer)
              u := segment(MatchBeginning(1)$Lisp+1,MatchEnd(1)$Lisp)$UniversalSegment(I
              getFile(s,headers.u)
          if reqtype = "POST" then
              StringMatch("command=(.*)$",headers)$Lisp
              u:UniversalSegment(Integer)
              u := segment(MatchBeginning(1)$Lisp+1,MatchEnd(1)$Lisp)$UniversalSegment(I
              getCommand(s,headers.u)
          WriteLine("multiServ end")$Lisp
          WriteLine("")$Lisp

getFile(s:SExpression,pathvar:String):Void ==
    WriteLine("")$Lisp
    WriteLine("getFile")$Lisp
    if not null? PATHNAME_-NAME(PATHNAME(pathvar)$Lisp)$Lisp then
    -- display contents of file
    --first determine Content-Type from file extension
        contentType:String := getContentType(pathvar)
        q:=Open(pathvar)$Lisp
        if null? q then
          q := MAKE_-STRING_-INPUT_-STREAM("File doesn't exist")$Lisp
          WriteLine("File does not exist.")$Lisp
    else
        q:=MAKE_-STRING_-INPUT_-STREAM("Problem with file path")$Lisp
    file:String := ""
    WriteLine("begin reading file")$Lisp
    r := MAKE_-STRING_-OUTPUT_-STREAM()$Lisp
    SiCopyStream(q,r)$Lisp
    filestream:String := GET_-OUTPUT_-STREAM_-STRING(r)$Lisp
    CLOSE(r)$Lisp
    CLOSE(q)$Lisp
    WriteLine("end reading file")$Lisp
    filelength:String := string(#filestream)
    file := concat ["Content-Length: ",filelength,STRING(NewLine$Lisp)$Lisp,STRING(N
    file := concat ["Connection: close",STRING(NewLine$Lisp)$Lisp,file]
    file := concat ["Content-Type: ",contentType,STRING(NewLine$Lisp)$Lisp,file]
    file := concat ["HTTP/1.1 200 OK",STRING(NewLine$Lisp)$Lisp,file]
```

```
      file := concat [file,filestream]
      f:=MAKE_-STRING_-INPUT_-STREAM(file)$Lisp
      SiCopyStream(f,s)$Lisp
      CLOSE(f)$Lisp
      CLOSE(s)$Lisp
      WriteLine("getFile end")$Lisp
      WriteLine("")$Lisp

  getCommand(s:SExpression,command:String):Void ==
      WriteLine$Lisp concat ["getCommand: ",command]
      SETQ(tmpmathml$Lisp, MAKE_-STRING_-OUTPUT_-STREAM()$Lisp)$Lisp
      SETQ(tmpalgebra$Lisp, MAKE_-STRING_-OUTPUT_-STREAM()$Lisp)$Lisp
      SETQ(savemathml$Lisp, _$texOutputStream$Lisp)$Lisp
      SETQ(savealgebra$Lisp, _$algebraOutputStream$Lisp)$Lisp
      SETQ(_$texOutputStream$Lisp,tmpmathml$Lisp)$Lisp
      SETQ(_$algebraOutputStream$Lisp,tmpalgebra$Lisp)$Lisp
--      parseAndInterpret$Lisp command
--      parseAndEvalStr$Lisp command
-- The previous two commands don't exit nicely when a syntactically incorrect command i
-- given to them.  They somehow need to be wrapped in CATCH statements but I haven't
-- figured out how to do this.  parseAndEvalToStringEqNum  uses the following CATCH
-- statements to call parseAndEvalStr but when I try these they don't work.  I get a
-- "NIL is not a valid identifier to use in AXIOM" message. Using parseAndEvalToString
-- works and doesn't crash on a syntax error.
--        v := CATCH('SPAD__READER, CATCH('top__level, parseAndEvalStr$Lisp command)$L
--        v = 'restart => ['"error"]
      ans := string parseAndEvalToStringEqNum$Lisp command
      SETQ(resultmathml$Lisp,GET_-OUTPUT_-STREAM_-STRING(_$texOutputStream$Lisp)$Lisp)
      SETQ(resultalgebra$Lisp,GET_-OUTPUT_-STREAM_-STRING(_$algebraOutputStream$Lisp)$
      SETQ(_$texOutputStream$Lisp,savemathml$Lisp)$Lisp
      SETQ(_$algebraOutputStream$Lisp,savealgebra$Lisp)$Lisp
      CLOSE(tmpmathml$Lisp)$Lisp
      CLOSE(tmpalgebra$Lisp)$Lisp
      -- Since strings returned from axiom are going to be displayed in html I
      -- should really check for the characters &,<,> and replace them with
      -- &amp;,&lt;,&gt;.  At present I only check for ampersands in formatMessages.
      mathml:String := string(resultmathml$Lisp)
      algebra:String := string(resultalgebra$Lisp)
      algebra := formatMessages(algebra)
      -- At this point mathml contains the mathml for the output but does not
      -- include step number or type information.  We should also save the command.
      -- I get the type and step number from the $internalHistoryTable
--       axans:String := concat ["<div><div class=_"command_">(",lastStep(),") -> ",com
      axans:String := concat ["<div class=_"stepnum_">", lastStep(), "</div><div class=
      WriteLine$Lisp concat ["mathml answer: ",mathml]
      WriteLine$Lisp concat ["algebra answer: ",algebra]
```

```
    q:=MAKE_-STRING_-INPUT_-STREAM(axans)$Lisp
    SiCopyStream(q,s)$Lisp
    CLOSE(q)$Lisp
    CLOSE(s)$Lisp


lastType():String ==
-- to examine the $internalHistoryTable uncomment the following lines
    WriteLine$Lisp "lastType begin"
--  WriteLine$Lisp string _$internalHistoryTable$Lisp
-- need to pick out first member of internalHistoryTable and then pick out
-- the element with % as first element, here's an example showing just
-- the first element of the list, which correponds to the last command.
-- Note that the last command does not necessarily correspond to the last
-- element of the first element of $internalHistoryTable as it is in this
-- example.
--(
-- (4 NIL
-- (x (value (BasicOperator) WRAPPED . #<vector 09a93bd0>))
-- (y (value (BasicOperator) WRAPPED . #<vector 09a93bb4>))
-- (% (value (Matrix (Polynomial (Integer))) WRAPPED . #<vector 0982e0e0>))
-- )
--...
--)
-- Also need to check for input error in which case the $internalHistoryTable
-- is not changed and the type retrieved would be that for the last correct
-- input.
    SETQ(first$Lisp,FIRST(_$internalHistoryTable$Lisp)$Lisp)$Lisp
    count:Integer := 0
    hisLength:Integer := LIST_-LENGTH(_$internalHistoryTable$Lisp)$Lisp
    length:Integer := LIST_-LENGTH(first$Lisp)$Lisp
    -- This initializes stepSav.  The test is a bit of a hack, maybe I'll
    -- figure out the right way to do it later.
    if string stepSav$Lisp = "#<OBJNULL>" then SETQ(stepSav$Lisp, 0$Lisp)$Lisp
    -- If hisLength = 0 then the history table has been reset to NIL
    -- and we're starting numbering over
    if hisLength = 0 then SETQ(stepSav$Lisp, 0$Lisp)$Lisp
    if hisLength > 0 and
      CAR(CAR(_$internalHistoryTable$Lisp)$Lisp)$Lisp ^= stepSav$Lisp then
        SETQ(stepSav$Lisp, CAR(CAR(_$internalHistoryTable$Lisp)$Lisp)$Lisp)$Lisp
        while count < length  repeat
            position(char "%",string FIRST(first$Lisp)$Lisp) = 2 => count := length+
            count := count +1
            SETQ(first$Lisp,REST(first$Lisp)$Lisp)$Lisp
    count = length + 1 => string SECOND(SECOND(FIRST(first$Lisp)$Lisp)$Lisp)$Lisp
    ""
```

```
  lastStep():String ==
      string CAR(CAR(_$internalHistoryTable$Lisp)$Lisp)$Lisp


  formatMessages(str:String):String ==
      WriteLine("formatMessages")$Lisp
      -- I need to replace any ampersands with &amp; and may also need to
      -- replace < and > with &lt; and &gt;
      strlist:List String
      WriteLine$Lisp "formatMessages1"
      WriteLine(str)$Lisp
      strlist := split(str,char "&")
      str := ""
      -- oops, if & is the last character in the string this method
      -- will eliminate it.  Need to redo this.
      for s in strlist repeat
          str := concat [str,s,"&amp;"]
      strlen:Integer := #str
      str := str.(1..(#str - 5))
      WriteLine$Lisp "formatMessages2"
--        WriteLine(str)$Lisp
      -- Here I split the string into lines and put each line in a "div".
      WriteLine$Lisp "formatMessages2.1"
      strlist := split(str, char string NewLine$Lisp)
      WriteLine$Lisp "formatMessages3"
      str := ""
      WriteLine("formatMessages4")$Lisp
      WriteLine(concat strlist)$Lisp
      for s in strlist repeat
          WriteLine(s)$Lisp
          str := concat [str,"<div>",s,"</div>"]
      WriteLine("formatMessages5")$Lisp

      str

  getContentType(pathvar:String):String ==
      WriteLine("getContentType begin")$Lisp
      -- set default content type
      contentType:String := "text/plain"
      -- need to test for successful match?
      StringMatch(".*\.(.*)$", pathvar)$Lisp
      u:UniversalSegment(Integer)
      u := segment(MatchBeginning(1)$Lisp+1,MatchEnd(1)$Lisp)$UniversalSegment(Integer
      extension:String := pathvar.u
      WriteLine$Lisp concat ["file extension: ",extension]
```

```
-- test for extensions: html, htm, xml, xhtml, js, css
if extension = "html" then
    contentType:String := "text/html"
else if extension = "htm" then
    contentType:String := "text/html"
else if extension = "xml" then
    contentType:String := "text/xml"
else if extension = "xhtml" then
    contentType:String := "application/xhtml+xml"
else if extension = "js" then
    contentType:String := "text/javascript"
else if extension = "css" then
    contentType:String := "text/css"
else if extension = "png" then
    contentType:String := "image/png"
else if extension = "jpg" then
    contentType:String := "image/jpeg"
else if extension = "jpeg" then
    contentType:String := "image/jpeg"
WriteLine$Lisp concat ["Content-Type: ",contentType]
WriteLine("getContentType end")$Lisp
contentType
```

3 Running Axiom Server

Put the extracted files in a suitable directory, like the one you started Axiom from, and issue the commands:

)set output mathml on
)lisp (load "http.lisp")
)compile axserver
axServer(8085,multiServ$AXSERV)

Of course you need a mathml enabled build of axiom to do this. You may also want to issue the command

)set messages autoload off
before starting the Axiom server.
or you can run inside axiom:
)read axserver.input

3.1 axserver.input

⟨*axserver.input*⟩≡
```
)set mes auto off
)set out mathml on
)lisp (load "http.lisp")
)compile axserver.spad
axServer(8085, multiServ)$AXSERV
```

4 License

⟨*license*⟩≡

⟨***⟩≡

  ⟨*license*⟩

  ⟨*package AXSERV AxiomServer*⟩

References

[1] nothing