An interpreter/translator for a dialect of John Backus’ FP language
Extension for Chicken Scheme
Version 1.1
# Table of Contents

## 1 About this egg
1.1 Version history ................................. 1  
1.2 Requirements .................................. 1  
1.3 Usage ........................................ 1  

## 2 Documentation
2.1 Introduction .................................... 2  
2.2 Objects ....................................... 2  
2.3 Built-in functions ............................. 2  
2.4 Grammar ...................................... 3  
2.5 Example ....................................... 4  
2.6 API .......................................... 4  
2.7 Interfacing to/from Scheme ................... 5  
2.8 Standard library ............................... 5  

## Index ........................................... 6
1 About this egg

1.1 Version history

1.1 Added some builtin functions, extended atom syntax
1.0 Initial release

1.2 Requirements

This egg requires the following extensions:
   silex, lalr

1.3 Usage

Load this egg like so:
   (require-extension fp)
2 Documentation

2.1 Introduction

This extension translates programs in a dialect of the FP programming language into Scheme. You can use the translator interactively, as a library or as a compiler extension (the latter allows you to compile FP code into executables and/or libraries).

To use it interactively, invoke the `fp-repl` procedure (see below). To use it as a library, call `fp-eval`.

If you want to compile FP programs, pass the `-X fp` option to the CHICKEN compiler driver, like this:

```plaintext
% csc -X fp myprogram.fp
```

A program consists of a list of definitions separated by semicolons, like this:

```plaintext
square == x [id, ~2];
main == square tonum
```

The left hand side of a definition specifies a name and the right hand side should be a functional form. A definition may be followed by auxiliary definitions enclosed in `{ ... }` which are only visible in the preceding definition.

Identifiers may consist of lowercase letters or an underscore. Any character with an ASCII/ISO-8859-1 code below or equal 32 is ignored. Any other character is treated as an identifier of length 1.

Comments follow C-style (`/* ... */`) and may not be nested. `#!` is also parsed as a comment and ignores everything up to the next line.

2.2 Objects

An object is an atom (a symbol consisting of uppercase characters or _, a character (`char`) a sequence (`<x1, ...>`), a character sequence " ... " or a number. The atom `F` is also used as the boolean false value. Atoms may also be given as `| ... |` when they should contain special characters.

If the `numbers` extension is loaded, then FP programs are capable of calculating with bignums and exact rationals.

2.3 Built-in functions

```plaintext
(f -> g; h): x  if f:x then g:x else h:x  [conditional]
(f g): x        f:(g:x)            [composition]
f x            f:x                [application]
/f:<x1, x2, ...>  f:<x1, f:<x2, ...>>  [insertion]
@f:<x1, x2, ...>  f:f:x, f:f:x2, ...>  [mapping]
N:<x1, x2, ...>  xN (negative number select from the right)
OP:<x, y>       x OP y, where OP is "+", "+", "+", (multiply), ", " (divide), "bior" (bitwise or), "band" (bitwise
and), or "bxor" (bitwise xor)

\begin{itemize}
\item \texttt{bnot}:x \quad \text{bitwise not}
\item \texttt{(~0)}:x \quad 0, where 0 is an object \hspace{1cm} \text{[constant]}
\item \texttt{[f1, f2, ...]}:x \quad <f1:x, f2:x, ...> \hspace{1cm} \text{[construction]}
\item \texttt{id}:x \quad x \hspace{1cm} \text{[identity]}
\item \texttt{hd}:<x1, x2, ...> \quad x1
\item \texttt{tl}:<x1, x2, ...> \quad <x2, ...>
\item \texttt{null}:x \quad \text{if} x = <> \text{ then T else F}
\item \texttt{atom}:x \quad \text{if} x = <...> \text{ then F else T}
\item \texttt{apndl}:<x, <y1, ...>> \quad \text{<x, y1, ...>}
\item \texttt{apndr}:<<x1, ...>, y> \quad \text{<x1, ..., y>}
\item \texttt{cat}:<<x1, ...>, <y1, ...>> \quad \text{<x1, ..., y1, ...>}
\item \texttt{reverse}:<x1, x2, ...> \quad \text{<xN, ..., x2, x1>}
\item \texttt{length}:<x1, ..., xN> \quad N
\item \texttt{eq}:<x, y> \quad \text{if} x = y \text{ then T else F}
\item \texttt{lt}:<x, y> \quad \text{less than (numbers only)}
\item \texttt{gt}:<x, y> \quad \text{greater than (numbers only)}
\item \texttt{not}:x \quad \text{if} x != F \text{ then T else F}
\item \texttt{tonum}:x \quad \text{converts character sequence to number}
\item \texttt{tochar}:x \quad \text{converts number to char}
\item \texttt{tostring}:x \quad \text{converts number into character sequence}
\item \texttt{tosym}:x \quad \text{converts character sequence to atom}
\item \texttt{(error "..."}):x \quad \text{prints error message and argument and exits}
\item \texttt{(debug "..."}):x \quad \text{prints debug message and argument}
\item \texttt{show}:x \quad \text{prints x and returns it}
\item \texttt{read}:s \quad \text{read contents for file with the name s}
\item \texttt{write}:<s1, s2> \quad \text{write string s2 into file with the name s1}
\item \texttt{(*f)}:<x1, x2, ...> \quad \text{removes elements from the sequence for which f:<x1} \text{ is false}
\item \texttt{(bu f x)}:y \quad f:<y, x>, x must be an object
\item ?:n \quad \text{returns a random integer between 0 and x-1}
\item \texttt{system}:s \quad \text{execute shell command and return status code}
\item \texttt{load}:s \quad \text{load FP source code or compiled .so/.dll}
\item \texttt{(while p f)}:x \quad \text{if p(x) is true, (while p f) : f(x), otherwise p(x)}
\item \texttt{and}:<x, y> \quad \text{if} x != F \text{ then y else F}
\item \texttt{or}:<x, y> \quad \text{if} x != F \text{ then x else y}
\item \texttt{f & g} \quad f \rightarrow g; \sim F
\item \texttt{f \sim n} \quad ff... (n times)
\item \texttt{gensym:symbol} \quad \text{return fresh symbol with name "xN", where "N" is some number}
\end{itemize}

(Alternative symbols are "." for "" and ">%" for ")"

### 2.4 Grammar

\begin{align*}
\text{PROGRAM} & \rightarrow \text{DEFINITION} \mid \text{APPLICATION} \ldots \\
\text{DEFINITION} & \rightarrow \text{ID} \ "==" \ \text{EXPR} \ ["(" \ \text{PROGRAM} \ ")"] \ [";\]}
\end{align*}
2.5 Example

/* fac.fp */

fac == eq0 -> ~1; x [id, fac sub1]
   { eq0 == eq [id, ~0];
     sub1 == - [id, ~1] }

main == fac tonum

2.6 API

fp-parse [procedure]
   (fp-parse INPUT)
Parses the FP code given in INPUT, which should be a string or an input port and returns its Scheme representation as a list of Scheme toplevel expressions. This Scheme code can be directly evaluated.

\[
\text{fp-eval} \quad \text{[procedure]}
\]

\[
\begin{align*}
\text{(fp-eval INPUT)} & \\
\text{Parses and evaluates the FP code given in INPUT.}
\end{align*}
\]

\[
\text{fp-repl} \quad \text{[procedure]}
\]

\[
\begin{align*}
\text{(fp-repl [PROMPT])} & \\
\text{Executes a read-eval-print-loop that prints PROMPT, reads a line of FP code and evaluates it, printing the returned result.}
\end{align*}
\]

2.7 Interfacing to/from Scheme

All top-level definitions in FP will result in a Scheme procedure definition of a procedure of one argument, with the name prefixed with \text{fp:}, so for example

\[
\text{fac} \equiv /x !
\]

will result in a procedure named \text{fp:fac} that you can call from Scheme like any other procedure.

FP programs can call Scheme procedures, provided they have a name with the \text{fp:} prefix and accept a single argument, returning a single value and accept/return values that are meaningful in FP programs. Scheme and FP data types are related in the following manner:

<table>
<thead>
<tr>
<th>Scheme</th>
<th>FP</th>
</tr>
</thead>
<tbody>
<tr>
<td>symbol</td>
<td>atom</td>
</tr>
<tr>
<td>char</td>
<td>char</td>
</tr>
<tr>
<td>list or string</td>
<td>sequence</td>
</tr>
<tr>
<td>number</td>
<td>number</td>
</tr>
</tbody>
</table>

2.8 Standard library

A small library of useful functions is installed in the CHICKEN extension repository under the name \text{stdlib.fp}, which you can access by putting \text{load:"stdlib.fp"} at the start of your FP program.
Index

fp-eval........................................... 5  fp-repl........................................... 5
fp-parse......................................... 4