Language Syntax

Webpage

We first have the syntax of the body of the webpage `<body>...</body>`,

```
bodyexp: anyHtmlCode> (<{exp}> anyHtmlCode)*
```

Expressions

General expressions:

```
exp: e, e', e'', ... ::=
    let <id> = e in e'
  | fun <id> -> e
  | fixfun <id> <id> -> e
  | e e'
  | if e then e' else e''
  | e;e'
  | <id>
  | <aexp> | <bexp> | <sexp>
  | <texp>
  | <uexp>
  | <html> | <dbexp>
  | (e)
  | begin e end
```

Arithmetic expressions:

```
aexp:
    <exp> + <exp>
  | <exp> - <exp>
  | <exp> * <exp>
  | <exp> / <exp>
  | <exp> ^ <exp>
  | <int literal>
```

Boolean expressions:

```
bexp:
    <exp> < <exp>
  | <exp> > <exp>
  | <exp> <= <exp>
  | <exp> >= <exp>
  | <exp> = <exp>
  | <exp> <> <exp>
  | <exp> && <exp>
  | <exp> || <exp>
  | not <exp>
  | <boolean literal>
```

String expressions:

```
sexp: <exp> ++ <exp> | <fstring literal>
```

Tuple expressions:

```
texp: fst <exp> | snd <exp> | <exp>, <exp>
```

Unit expression:

```
uexp: ()
```

TODO

HTML:

```
html: <[ anyHtmlCode ]>
```

Database expression:

```
dbexp: sqlite_opendb | sqlite_closedb | sqlite_exec
```

For the time being, only couples are allowed, and `(x1, x2, x3, x4)` is parsed as `(x1, (x2, (x3, x4)))`.

Identifiers (variable and function names)

```
(_|[a-z])(_|'|[0-9]|[a-z]|[A-Z])*
```

Examples:

- variable
- my_function_n_362
- _MyFunction
- myVariable067

But not:

- MyFunction
- 01var

Literals

Integers

For better readability for the programmer, we allow underscores in numbers.

```
[0-9]([0-9]|_)*
```

Examples:

- 123
- 100_000
- 1_2_____3____

Strings

Strings are delimited by quotes: `"..."`.

Format strings

Format strings are delimited by: `f"..."`. A formatter can be inserted in a format string with `%(value)` TODO not implemented *yet*.

Booleans

`true`, `false`

Namespacing

Modules' names starts with a capital letter but otherwise are the same as variable names.

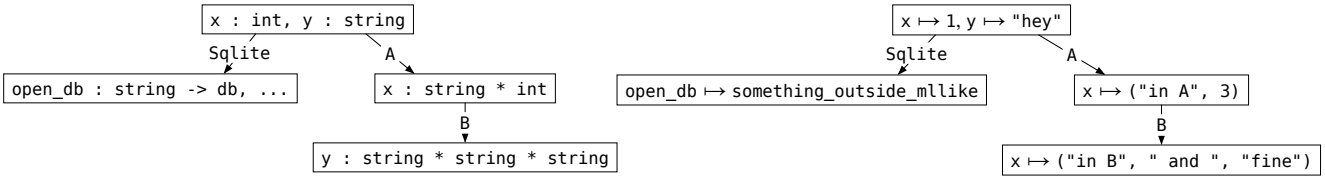
Variables are namespaced, more precisely:

A variable is a variable name preceded by the module where it's defined e.g. `Sqlite.exec`.

At some point, we would like to implement modules contained inside another module, maybe even functors if possible, and records types. When this is done, a full variable name will be `((modulevar.)*(expr.)*varname)` where each `expr` must be of type record.

For instance, if module A contains module B which itself contains a record `r : {r' : rec'}` where `rec'` itself is a record with a field x, then `A.B.r.r'.x` designate the field x of the farthest nested record.

A modular typing (resp. evaluation) environment therefore becomes a tree, where each edge is labelled with a module name, and each node is labelled with a typing (resp. evaluation) environment.



Remark. For now, user declared modules are not implemented, thus the only modules are Get, Post, and Sqlite.

Type system

Types

`<tlit> : int | bool | string | unit | html | db`

$\alpha, \beta, \ldots$ `::=` $\alpha \to \beta$ `|` $\alpha \times \beta$

Typing rules

$$\frac{\Gamma \vdash \texttt{e} : \alpha \quad \Gamma, \texttt{x} : \alpha \vdash \texttt{e'} : \beta}{\Gamma \vdash \texttt{let x = e in e'} : \beta} \qquad \frac{\Gamma, \texttt{x} : \alpha \vdash \texttt{e} : \beta}{\Gamma \vdash \texttt{fun x -> e} : \alpha \to \beta} \qquad \frac{\Gamma, \texttt{f} : \alpha \to \beta, \texttt{x} : \alpha \vdash \texttt{e} : \beta}{\Gamma \vdash \texttt{fixfun f x -> e} : \alpha \to \beta}$$

$$\frac{\Gamma \vdash \texttt{e} : \alpha \to \beta \quad \Gamma \vdash \texttt{e'} : \alpha}{\Gamma \vdash \texttt{e e'} : \beta} \qquad \frac{\Gamma \vdash \texttt{e} : \texttt{bool} \quad \Gamma \vdash \texttt{e'} : \alpha \quad \Gamma \vdash \texttt{e''} : \alpha}{\Gamma \vdash \texttt{if e then e' else e''} : \alpha} \qquad \frac{\Gamma \vdash \texttt{e} : \texttt{unit} \quad \Gamma \vdash \texttt{e'} : \alpha}{\Gamma \vdash \texttt{e;e'} : \alpha}$$

$$\circledast\texttt{: +, -, *, /, or ^} \ \frac{\Gamma \vdash \texttt{e} : \texttt{int} \quad \Gamma \vdash \texttt{e'} : \texttt{int}}{\Gamma \vdash \texttt{e} \ \circledast \ \texttt{e'} : \texttt{int}} \qquad \circledast\texttt{: >, <, >=, <=, = or <>} \ \frac{\Gamma \vdash \texttt{e} : \alpha \quad \Gamma \vdash \texttt{e'} : \alpha}{\Gamma \vdash \texttt{e} \ \circledast \ \texttt{e'} : \texttt{bool}}$$

$$\circledast\texttt{: \&\& or ||} \ \frac{\Gamma \vdash \texttt{e} : \texttt{bool} \quad \Gamma \vdash \texttt{e'} : \texttt{bool}}{\Gamma \vdash \texttt{e} \ \circledast \ \texttt{e'} : \texttt{bool}} \qquad \frac{\Gamma \vdash \texttt{e} : \texttt{bool}}{\Gamma \vdash \texttt{not e} : \texttt{bool}} \qquad \frac{\Gamma(x) = \alpha}{\Gamma \vdash x : \alpha}$$

$$\frac{}{\Gamma \vdash \texttt{b} : \texttt{bool}} \qquad \frac{}{\Gamma \vdash \texttt{n} : \texttt{int}} \qquad \frac{}{\Gamma \vdash \texttt{<(f)string literal>} : \texttt{string}} \qquad \frac{}{\Gamma \vdash \texttt{<[html code]>} : \texttt{html}}$$

$$\frac{\Gamma \vdash \texttt{e} : \alpha \quad \Gamma \vdash \texttt{e'} : \beta}{\Gamma \vdash \texttt{(e, e')} : \alpha \times \beta} \qquad \frac{}{\Gamma \vdash \texttt{fst} : \alpha \times \beta \to \alpha} \qquad \frac{}{\Gamma \vdash \texttt{snd e} : \alpha \times \beta \to \beta} \qquad \frac{}{\Gamma \vdash \texttt{()} : \texttt{unit}}$$

$$\frac{}{\Gamma \vdash \texttt{sqlite\_opendb} : \texttt{string} \to \texttt{db}} \qquad \frac{}{\Gamma \vdash \texttt{sqlite\_closedb} : \texttt{db} \to \texttt{bool}}$$

$$\frac{}{\Gamma \vdash \texttt{sqlite\_exec} : \texttt{db} \to (\texttt{html} \to \texttt{html} \to \texttt{html}) \to (\texttt{html} \to \texttt{string} \to \texttt{string} \to \texttt{html}) \to \texttt{bool}}$$

Program semantics

Values

`values: v, v', ... ::=` ⟨`E, <function>`⟩ `| n | true | false | <string literal> | (v, v') | <vdb> | pure_html_code |` `evald_page`

`function: fun x -> e | fixfun x -> e`

`evald_page: [v1; v2; ...; vn]` ⚠️ it's a list in the meta-language.

`vdb: a value representing a database in the language`

An evaluated webpage can be injected in a value (via the frame). This happens when we evaluate, e.g.

`<{ <[htmlcode <{let x = 1 in x}> somemorehtmlcode]> }>`

Evaluation rules

A dynamic webpage to evaluate is seen as a list of either:

- Pure html code ;
- An expression ;
- A global declaration.

The top-level interpreted page is a dynamic webpage, as well as the content between HTML opening/closing bracket.

The interpreter evaluates following a big-step call-by-value semantics. We define two mutually recursive relations to evaluate expressions and dynamic webpages.

Expression

$$\frac{}{E \vdash n \Downarrow n} \quad \frac{}{E \vdash \texttt{true} \Downarrow \texttt{true}} \quad \frac{}{E \vdash \texttt{false} \Downarrow \texttt{false}} \quad \frac{}{E \vdash \texttt{"<string>"} \Downarrow \texttt{"<string>"}}$$

$$\frac{}{E \vdash \texttt{fun x -> e} \Downarrow \langle E, \texttt{fixfun f x -> e} \rangle} \quad \frac{}{E \vdash \texttt{fixfun f x -> e} \Downarrow \langle E, \texttt{fun x -> e} \rangle}$$

$$\frac{E \vdash \texttt{e} \Downarrow \texttt{v} \quad E \vdash \texttt{e'} \Downarrow \texttt{v'}}{E \vdash \texttt{(e, e')} \Downarrow \texttt{(v, v')}} \quad \frac{E \vdash \texttt{e} \Downarrow n \quad E \vdash \texttt{e'} \Downarrow m}{E \vdash \texttt{e} \circledast \texttt{e'} \Downarrow n \mathbin{\overline{\circledast}} n'} \quad \frac{E \vdash \texttt{e} \Downarrow \texttt{n}}{E \vdash \texttt{-e} \Downarrow -n} \quad \frac{E \vdash \texttt{e} \Downarrow \texttt{b} \quad E \vdash \texttt{e'} \Downarrow \texttt{b'}}{E \vdash \texttt{e} \circledast \texttt{e'} \Downarrow b \mathbin{\overline{\circledast}} b'}$$

$$\frac{E \vdash \texttt{e} \Downarrow \texttt{b}}{E \vdash \texttt{not e} \Downarrow \neg b} \quad \frac{E \vdash \texttt{e} \Downarrow \texttt{s} \quad E \vdash \texttt{e'} \Downarrow \texttt{s'}}{E \vdash \texttt{e} \mathbin{+\!\!+} \texttt{e'} \Downarrow s +\!\!+ s'}$$

$$\frac{E \vdash \texttt{e} \Downarrow \langle \texttt{E', fun x -> e\_f} \rangle \quad E \vdash \texttt{e'} \Downarrow \texttt{v} \quad \texttt{E', x} \mapsto \texttt{v} \vdash \texttt{e\_f} \Downarrow \texttt{v'}}{E \vdash \texttt{e e'} \Downarrow \texttt{v'}} \quad \frac{E \vdash \texttt{e'} \Downarrow \texttt{v'} \quad \texttt{E, x} \mapsto \texttt{v'} \vdash \texttt{e'} \Downarrow \texttt{v}}{E \vdash \texttt{let x = e in e'} \Downarrow \texttt{v}}$$

$$\frac{E \vdash \texttt{e} \Downarrow \langle \texttt{E', fixfun f x -> e\_f} \rangle \quad E \vdash \texttt{e'} \Downarrow \texttt{v} \quad \texttt{E, f} \mapsto \texttt{fixfun f x -> e\_f, x} \mapsto \texttt{v} \vdash \texttt{e\_f} \Downarrow \texttt{v'}}{E \vdash \texttt{e e'} \Downarrow \texttt{v'}}$$

$$\frac{E \vdash \texttt{e} \Downarrow \texttt{v} \quad E \vdash \texttt{e'} \Downarrow \texttt{v'}}{E \vdash \texttt{e;e'} \Downarrow \texttt{v'}} \quad \frac{E \vdash \texttt{e} \Downarrow \texttt{true} \quad E \vdash \texttt{e'} \Downarrow \texttt{v'}}{E \vdash \texttt{if e then e' else e''} \Downarrow \texttt{v'}} \quad \frac{E \vdash \texttt{e} \Downarrow \texttt{false} \quad E \vdash \texttt{e''} \Downarrow \texttt{v''}}{E \vdash \texttt{if e then e' else e''} \Downarrow \texttt{v''}}$$

$$\frac{E \vdash \texttt{e} \Downarrow \texttt{(v, v')}}{E \vdash \texttt{fst e} \Downarrow \texttt{v}} \quad \frac{E \vdash \texttt{e} \Downarrow \texttt{(v, v')}}{E \vdash \texttt{snd e} \Downarrow \texttt{v'}} \quad \texttt{E(x) = v} \frac{E \vdash \texttt{e} \Downarrow \texttt{x}}{E \vdash \texttt{e} \Downarrow \texttt{v}} \quad \frac{E \vdash \texttt{dynamic\_webpage} \Downarrow \texttt{evald\_page}}{E \vdash \texttt{<[dynamic\_webpage]>} \Downarrow \boxed{\texttt{evald\_page}}}$$

vdb is a projection within the langhage of the db at path s $\dfrac{E \vdash \texttt{e} \Downarrow \texttt{s}}{E \vdash \texttt{sqlite\_opendb e} \Downarrow \texttt{vdb}}$

If the corresponding db could be closed $\dfrac{E \vdash \texttt{e} \Downarrow \texttt{vdb}}{E \vdash \texttt{sqlite\_closedb e} \Downarrow \texttt{true}}$

If the corresponding db couldn't be closed (it remains oen) $\dfrac{E \vdash \texttt{e} \Downarrow \texttt{vdb}}{E \vdash \texttt{sqlite\_closedb e} \Downarrow \texttt{true}}$

$$\frac{E \vdash \texttt{e} \Downarrow \texttt{vdb} \quad E \vdash \text{a closure for function } \texttt{fl} \Downarrow \texttt{vdb} \quad E \vdash \text{a closure for function } \texttt{fc} \Downarrow \texttt{vdb} \quad E \vdash \texttt{e'} \Downarrow \texttt{s}}{E \vdash \texttt{sqlite\_exec e f1 f2 e'} \Downarrow \texttt{processed s*}}$$

*The semantic of exec is as follows: `s` is a string corresponding to a SQL command, which is executed on the database `vdb`. If it has query statements, then `sqlite_exec` allow to process the result with a double fold left function applied on the resulting table i.e. let:

header_1	...	header_n
data_1	...	data_n

be a line of the resulting table of the query. We first allow to process one line in the following manner: the value `line_i` corresponding to the table above is: `fc (... (fc EmptyHtmlCode header_1 data_1) ...) header_n data_n`.

Similarly, the result of each line is combined in the followind manner to form `processed s`:
`processed s = fl (... (fl EmptyHtmlCode line_1) ...) line_n`.

Actually, `fl` and `fc` can't be any functions for weird reasons. TODO explain or fix + see if it leads to actual limitations.

See here for reason the db couldn't be closed, and more specifications of the sqlite functions.

Dynamic webpage

$$\frac{\texttt{E} \vdash \texttt{e} \Downarrow \texttt{v} \quad \texttt{E, x} \mapsto \texttt{v} \vdash \texttt{page} \Downarrow \texttt{evald}}{\texttt{E} \vdash \texttt{(let x = e) :: page} \Downarrow \texttt{evald}} \qquad \frac{\texttt{E} \vdash \texttt{page} \Downarrow \texttt{evald}}{\texttt{E} \vdash \texttt{pure\_html :: page} \Downarrow \texttt{pure\_html :: evald}}$$

$$\frac{\texttt{E} \vdash \texttt{e} \Downarrow \texttt{v} \quad \texttt{E} \vdash \texttt{page} \Downarrow \texttt{evald}}{\texttt{E} \vdash \texttt{e :: page} \Downarrow \texttt{v :: evald}}$$

TODO write documentation for session variables

TODO

□ Implement fstrings

☑️ Implement percent-encoding

□ Don’t lex ml located in html comment

□ Add comments within ML

☑️ Allow HTML brackets to contain any dynpage e.g. <[somehtml <{“coucou”}> somemorehtml]>

□ Add syntactic sugar for multiple variables functions.

□ Add t-uples

□ Add pattern-matching

☑️ Add superglobal variables

☑️ Add user-defined global variables

□ At some point, we would like to implement modules contained inside another module, and records types.

□ Add user-defined types

□ Once it’s done, implement basic types such as list directly within the language.

□ Allow type annotations from the user

□ Allow importing other ml files (as modules ?)

□ Keep line number information on parsed term for better typing error messages (?)