

Language Syntax

Webpage

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

bodyexp: <any html code> | <any html code> <{exp}> <bodyexp>

Example:

```
<body>
  <h1>Exampel</h1>
  <{
    let x = 1 in
    if x = 2 then
  }>
  2
  <{
    else
  }>
  what?
  <{}>
</body>
```

Expressions

General expressions:

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

HTML:

```
html: }> any html code <{
```

For now, only couples are allowed, and (x_1, x_2, x_3, x_4) is parsed as $(x_1, (x_2, (x_3, x_4)))$.

Identifiers (variable and function names)

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

Examples:

- variable
- my_function
- _MyFunction
- myVariable

But not:

- MyFunction
- 01var

Literals

Integers

For 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})`

Booleans

true, false

Type system

Types

$\text{<lit>} ::= \text{int} \mid \text{bool} \mid \text{string} \mid \text{unit} \mid \text{html}$

$$\begin{array}{c}
 \frac{\Gamma \vdash e : \alpha \quad \Gamma, x : \alpha \vdash e' : \beta}{\Gamma \vdash \text{let } x = e \text{ in } e' : \beta} \quad \frac{\Gamma, x : \alpha \vdash e : \beta}{\Gamma \vdash \text{fun } x \rightarrow e : \alpha \rightarrow \beta} \quad \frac{\Gamma, f : \alpha \rightarrow \beta, x : \alpha \vdash e : \beta}{\Gamma \vdash \text{fixfun } f \ x \rightarrow e : \alpha \rightarrow \beta} \\
 \frac{\Gamma \vdash e : \alpha \rightarrow \beta \quad \Gamma \vdash e' : \alpha}{\Gamma \vdash e \ e' : \beta} \quad \frac{\Gamma \vdash e : \text{bool} \quad \Gamma \vdash e' : \alpha \quad \Gamma \vdash e'' : \alpha}{\Gamma \vdash \text{if } e \text{ then } e' \text{ else } e'' : \alpha} \quad \frac{\Gamma \vdash e : \text{unit} \quad \Gamma \vdash e' : \alpha}{\Gamma \vdash e; e' : \alpha} \\
 \frac{\otimes: +, -, *, /, \text{or}^{\wedge} \quad \frac{\Gamma \vdash e : \text{int} \quad \Gamma \vdash e' : \text{int}}{\Gamma \vdash e \otimes e' : \text{int}} \quad \otimes: >, <, \geq, \leq, = \text{or} \Leftrightarrow \frac{\Gamma \vdash e : \alpha \quad \Gamma \vdash e' : \alpha}{\Gamma \vdash e \otimes e' : \text{bool}}}{\Gamma \vdash e \otimes e' : \text{bool}} \\
 \frac{\otimes: \& \text{ or} \mid \mid \quad \frac{\Gamma \vdash e : \text{bool} \quad \Gamma \vdash e' : \text{bool}}{\Gamma \vdash e \otimes e' : \text{bool}} \quad \frac{\Gamma \vdash e : \text{bool}}{\Gamma \vdash \text{not } e : \text{bool}}}{\Gamma(x) = \alpha \quad \frac{}{\Gamma \vdash x : \alpha}}
 \end{array}$$

$$\Gamma \vdash b : \text{bool} \quad \Gamma \vdash n : \text{int} \quad \Gamma \vdash \text{<string literal>} : \text{string} \quad \Gamma \vdash \text{<fstring literal>} : \text{string}$$

TODO

- c.f. example at the beginning of the document, we want to be able to consider `if b then }> ... <{ else }> ...` as a valid if-then-else, although `<{f}> ...` shouldn't be understood as the application of f to something... Or should it ? Investigate.
- Add syntactic sugar for multiple variables functions.
- Add t-uples
- Add pattern-matching
- Add global from outside the function variables
- Add user-defined global variables
- Add user-defined types
- Once it's done, implement basic types such as list directly within the language.
- Maybe revisit sequence's semantics. We may want `<{"<tag>" ; "hey</tag>"}>` to produce the html code `<tag>hey</tag>`.