

4 Liquid types for Elm

4.1 Formal definition of Elm

4.1.1 Syntax

Elm differentiates variables depending on the capitalization of the first letter. For the formal language we define $\langle \text{upper-var} \rangle \in \mathcal{T}$ for variables with the first letter capitalized and $\langle \text{lower-var} \rangle \in \mathcal{T}$ for variables without.

Syntactically we can build our types from booleans, integers, lists, tuples, records, functions, custom types and type variables.

We will define our syntax in a Backus-Naur-Form [Bac59].

Definition 4.1: Type Signiture Syntax

We define the following types:

```

$$\begin{aligned} \langle \text{upper-var} \rangle &\in \mathcal{T} \\ \langle \text{lower-var} \rangle &\in \mathcal{T} \\ \langle \text{list-lower-var} \rangle &::= "" \mid \langle \text{lower-var} \rangle \mid \langle \text{lower-var} \rangle \langle \text{list-lower-var} \rangle \\ \langle \text{list-type-fields} \rangle &::= "" \mid \langle \text{lower-var} \rangle ":" \langle \text{type} \rangle \\ &\quad \mid \langle \text{lower-var} \rangle ":" \langle \text{type} \rangle "," \langle \text{list-type-fields} \rangle \\ \langle \text{list-type} \rangle &::= "" \mid \langle \text{type} \rangle \mid \langle \text{type} \rangle \langle \text{list-type} \rangle \\ \langle \text{type} \rangle &::= \text{"Bool"} \\ &\quad \mid \text{"Int"} \\ &\quad \mid \text{"List"} \langle \text{type} \rangle \\ &\quad \mid \text{"("} \langle \text{type} \rangle "," \langle \text{type} \rangle \text{"}")} \\ &\quad \mid \text{" " } \langle \text{list-type-fields} \rangle \text{" " } \\ &\quad \mid \langle \text{type} \rangle \text{"->" } \langle \text{type} \rangle \\ &\quad \mid \langle \text{upper-var} \rangle \langle \text{list-type} \rangle \\ &\quad \mid \langle \text{lower-var} \rangle \end{aligned}$$

```

For matching expressions we allow various pattern.

Definition 4.2: Pattern Syntax

We define the following types:

```

$$\begin{aligned} \langle \text{list-pattern-list} \rangle &::= "" \mid \langle \text{pattern} \rangle \\ &\quad \mid \langle \text{pattern} \rangle "," \langle \text{list-pattern-list} \rangle \\ \langle \text{list-pattern-sort} \rangle &::= "" \mid \langle \text{pattern} \rangle \mid \langle \text{pattern} \rangle \langle \text{list-pattern-sort} \rangle \end{aligned}$$

```

```

<list-pattern-vars> ::= "" | <lower-var>
                      | <lower-var> ", " <list-pattern-vars>

<pattern> ::= <bool>
            | <int>
            | "[" <list-pattern-list> "]"
            | "(" <pattern> , <pattern> ")"
            | <upper-var> <list-pattern-sort>
            | <lower-var>
            | <pattern> "as" <lower-var>
            | "{" <list-pattern-vars> "}"
            | <pattern> "::" <pattern>
            | "-"

```

Because Elm is a pure functional programming language, a program contains just a single expression.

Definition 4.3: Expression Syntax

We define the following types:

```

<list-exp-field> ::= <lower-var> "=" <exp>
                  | <lower-var> "=" <exp> ", " <list-exp-field>

<maybe-signature> ::= "" | <lower-var> ":" <type> ";"

<list-case> ::= <pattern> "->" <exp>
              | <pattern> "->" <exp> ";" <list-case>

<bool> ::= "True" | "False"

<int> ::= "0" | "-1" | "1" | "-2" | "2" | ...

<list-exp> ::= "" | <exp> | <exp> ", " <list-exp>

```

The definition of <exp> can be found in figure 1.

Additionally, Elm also allows global constants, type aliases and custom types.

Definition 4.4: Statement Syntax

We define the following types:

```

<list-sort> ::= <upper-var> <list-type>
              | <upper-var> <list-type> | <list-sort>

```

```

<exp> ::= "foldl"
        | "(:)"
        | "(+)" | "(-)" | "(*)" | "(//)"
        | "<" | "(==)"
        | "not" | "&&" | "(||)"
        | <exp> ">" <exp>
        | <exp> ">>" <exp>
        | "if" <exp> "then" <exp> "else" <exp>
        | "{" <list-exp-field> "}"
        | "{}"
        | "{" <lower-var> "|" <list-exp-field> "}"
        | <lower-var> "." <lower-var>
        | "let" <maybe-signature> <lower-var> "=" <exp> "in" <exp>
        | "case" <exp> "of" "[" <list-case> "]"
        | <exp> <exp>
        | <bool>
        | <int>
        | "[" <list-exp> "]"
        | "(" <exp> "," <exp> ")"
        | "\" <pattern> "->" <exp>
        | <upper-var>
        | <lower-var>

```

Figure 1: Syntax for Expressions

Example 4.1

`foldl` iterates over the list from left to right. It takes the function `(: :)`, that appends an element to a list, and the empty list as the starting list. The `main` function reverses the list and returns the first element: 3. Elm requires you also provide return values for other cases that may occur, like the empty list. In that case we just return `-1`. This will never happen, as long as the reverse function is correctly implemented.

[Bac59] John W. Backus. “The syntax and semantics of the proposed international algebraic language of the Zurich ACM-GAMM Conference”. In: *IFIP Congress*. 1959, pp. 125–131.