

# Aurox language specification

Jakub Grobelny

21 maja 2019

## 1 Syntax

Character classification:

- $\langle whitespace \rangle$  — HT, LF, CR, SPACE
- $\langle digit \rangle$  — 0-9
- $\langle lowercase \rangle$  — underscore or any other lowercase Unicode characters<sup>1</sup>
- $\langle uppercase \rangle$  — any uppercase Unicode characters<sup>2</sup>
- $\langle special \rangle$  — '-', '+', '\*', '/', '=', '>', '<', '.', '!', '@', '%', '^, ' ', '&', '\$, '|'

Any character sequence beginning with character # ending with LF are *comments*.

$\langle operator \rangle ::= \langle special \rangle$   
|  $\langle special \rangle \langle operator \rangle$

$\langle keyword \rangle ::= \text{let} \mid \text{in} \mid \text{if} \mid \text{and} \mid \text{or} \mid \text{then}$   
| **match** | **else** | **with** | **type** | **import**  
| **define** | **\_** | **defop** | **end** | **case**

---

<sup>1</sup>All characters X which satisfy `char_type(X, lower)` predicate in SWI Prolog

<sup>2</sup>All characters X which satisfy `char_type(X, upper)` predicate in SWI Prolog

$\langle identifier \rangle ::= \langle lowercase \rangle \langle alphanum \rangle$   
 $\langle type\ name \rangle ::= \langle uppercase \rangle \langle alphanum \rangle$   
 $\langle alphanum \rangle ::= \langle alphanum\ char \rangle \langle alphanum \rangle \mid \epsilon$   
 $\langle alphanum\ char \rangle ::= \langle lowercase \rangle \mid ? \mid \langle digit \rangle$   
 $\langle integer \rangle ::= \langle digit \rangle \mid \langle digit \rangle \langle integer \rangle$   
 $\langle float \rangle ::= \langle integer \rangle . \langle digit\ sequence \rangle \langle exponent \rangle \mathbf{e}$   
 $\quad \mid \langle integer \rangle \langle exponent \rangle$   
 $\langle digit\ sequence \rangle ::= \langle digit \rangle \mid \langle digit \rangle \langle digit\ sequence \rangle$   
 $\langle e \rangle ::= \mathbf{e} \mid \mathbf{E}$   
 $\langle exponent \rangle ::= \langle e \rangle - \langle integer \rangle$   
 $\quad \mid \langle e \rangle \langle integer \rangle$   
 $\langle boolean \rangle ::= \mathbf{false} \mid \mathbf{true}$   
 $\langle string \rangle ::= \text{``} \langle char\ sequence \rangle \text{''} \mid \text{'''}$   
 $\langle char \rangle ::= \text{' } \langle character \rangle \text{'}$   
 $\langle char\ sequence \rangle ::= \langle character \rangle \mid \langle character \rangle \langle char\ sequence \rangle$   
 $\langle character \rangle ::= \text{Unicode} \mid \backslash \backslash \mid \backslash \mathbf{b} \mid \backslash \mathbf{n} \mid \backslash \mathbf{f} \mid \backslash \mathbf{a} \mid \backslash \mathbf{r} \mid \backslash \mathbf{t} \mid \backslash \mathbf{0} \mid \backslash \text{''} \mid \backslash \text{'}$   
 $\langle program \rangle ::= \langle operator\ declaration \rangle \langle program \rangle$   
 $\quad \mid \langle import \rangle \langle program \rangle$   
 $\quad \mid \langle expression\ sequence \rangle \langle program \rangle$   
 $\quad \mid \langle definition \rangle \langle program \rangle$   
 $\quad \mid \epsilon$   
 $\langle operator\ declaration \rangle ::= \mathbf{defop} \langle operator \rangle \langle integer \rangle \langle associativity \rangle$   
 $\langle associativity \rangle ::= \mathbf{left} \mid \mathbf{right} \mid \mathbf{none} \mid \mathbf{prefix} \mid \mathbf{postfix}$   
 $\langle import \rangle ::= \mathbf{import} \langle import\ list \rangle \mathbf{end}$

$$\begin{aligned}
\langle \text{import list} \rangle &::= \epsilon \mid \langle \text{string} \rangle \langle \text{import list} \rangle \\
&\quad \mid \langle \text{type name} \rangle \langle \text{import list} \rangle \\
\langle \text{definition} \rangle &::= \mathbf{define} \langle \text{function name} \rangle \langle \text{formal parameters} \rangle : \\
&\quad \langle \text{type} \rangle = \langle \text{expression sequence} \rangle \mathbf{end} \\
\langle \text{function name} \rangle &::= \langle \text{identifier} \rangle \mid ( \langle \text{operator} \rangle ) \\
\langle \text{formal parameters} \rangle &::= \langle \text{variable name} \rangle \langle \text{formal parameters} \rangle \mid \epsilon \\
\langle \text{variable name} \rangle &::= \langle \text{identifier} \rangle \mid \_ \\
\langle \text{type} \rangle &::= \langle \text{function type} \rangle \\
&\quad \mid \langle \text{function type} \rangle , \langle \text{tupe} \rangle \\
\langle \text{function type} \rangle &::= \langle \text{algebraic data type} \rangle \\
&\quad \mid \langle \text{function type} \rangle (->) \langle \text{algebraic data type} \rangle \\
\langle \text{algebraic data type} \rangle &::= \langle \text{type name} \rangle \langle \text{atomic type sequence} \rangle \\
&\quad \mid \langle \text{atomic type} \rangle \\
\langle \text{atomic type sequence} \rangle &::= \langle \text{atomic type} \rangle \langle \text{atomic type sequence} \rangle \mid \epsilon \\
\langle \text{atomic type} \rangle &::= \langle \text{identifier} \rangle \mid \langle \text{type name} \rangle \\
&\quad \mid [ \langle \text{type} \rangle ] \mid ( \langle \text{type} \rangle ) \\
\langle \text{type definition} \rangle &::= \mathbf{type} \langle \text{type name} \rangle \langle \text{formal parameters} \rangle \mathbf{with} \\
&\quad \langle \text{type constructors} \rangle \mathbf{end} \\
\langle \text{type constructors} \rangle &::= \mathbf{case} \langle \text{type name} \rangle \langle \text{atomic type} \rangle \\
&\quad \mid \mathbf{case} \langle \text{type name} \rangle \\
\langle \text{expression sequence} \rangle &::= \langle \text{expression} \rangle \\
&\quad \mid \langle \text{expression} \rangle ; \langle \text{expression sequence} \rangle \\
\langle \text{expression} \rangle &::= \langle \text{pattern matching} \rangle \mid \langle \text{let definition} \rangle \\
&\quad \mid \langle \text{conditional expression} \rangle \mid \langle \text{tuple expression} \rangle \\
\langle \text{let definition} \rangle &::= \mathbf{let} \langle \text{variable name} \rangle : \langle \text{type} \rangle = \\
&\quad \langle \text{expression sequence} \rangle \mathbf{in} \langle \text{expression sequence} \rangle \mathbf{end}
\end{aligned}$$

$\langle \text{conditional expression} \rangle ::= \text{if } \langle \text{expression sequence} \rangle \text{ then}$   
 $\quad \langle \text{expression sequence} \rangle \text{ else } \langle \text{expression sequence} \rangle \text{ end}$

$\langle \text{pattern matching} \rangle ::= \text{match } \langle \text{expression sequence} \rangle \text{ with}$   
 $\quad \langle \text{pattern matching cases} \rangle \text{ end}$

$\langle \text{pattern matching cases} \rangle ::= \langle \text{pattern case} \rangle \langle \text{pattern matching cases} \rangle$   
 $\quad | \quad \epsilon$

$\langle \text{pattern case} \rangle ::= \text{case } \langle \text{pattern} \rangle \Rightarrow \langle \text{expression sequence} \rangle$

$\langle \text{pattern} \rangle ::= \langle \text{deconstructor pattern} \rangle$   
 $\quad | \quad \langle \text{deconstructor pattern} \rangle , \langle \text{pattern} \rangle$

$\langle \text{deconstructor pattern} \rangle ::= \langle \text{type name} \rangle \langle \text{atomic pattern} \rangle$   
 $\quad | \quad \langle \text{atomic pattern} \rangle$

$\langle \text{atomic pattern} \rangle ::= \langle \text{variable name} \rangle | \langle \text{type name} \rangle$   
 $\quad | \quad ( \langle \text{pattern} \rangle )$   
 $\quad | \quad \langle \text{list pattern} \rangle$   
 $\quad | \quad \langle \text{constant} \rangle$

$\langle \text{list pattern} \rangle ::= [ \langle \text{pattern} \rangle | \langle \text{variable name} \rangle ]$   
 $\quad | \quad [ \langle \text{pattern} \rangle ]$   
 $\quad | \quad [ ]$

$\langle \text{constant} \rangle ::= \langle \text{integer} \rangle | \langle \text{boolean} \rangle | \langle \text{float} \rangle$   
 $\quad | \quad ( ) | \langle \text{string} \rangle | \langle \text{char} \rangle$

$\langle \text{tuple expression} \rangle ::= \langle \text{logical or} \rangle , \langle \text{tuple expression} \rangle$   
 $\quad | \quad \langle \text{logical or} \rangle$

$\langle \text{logical or} \rangle ::= \langle \text{logical and} \rangle \text{ and } \langle \text{logical or} \rangle$   
 $\quad | \quad \langle \text{logical and} \rangle$

$\langle \text{logical and} \rangle ::= \langle \text{expression none } 0 \rangle \text{ and } \langle \text{logical and} \rangle$   
 $\quad | \quad \langle \text{expression none } 0 \rangle$

$\langle \text{expression none } N \rangle ::= \langle \text{expression right } N \rangle \langle \text{operator none } N \rangle$   
 $\quad \langle \text{expression none } N \rangle$   
 $\quad | \quad \langle \text{expression right } N \rangle$

$$\begin{aligned}
\langle \textit{expression right } N \rangle &::= \langle \textit{expression left } N \rangle \langle \textit{operator right } N \rangle \\
&\quad \langle \textit{expression right } N \rangle \\
&\quad | \quad \langle \textit{expression left } N \rangle \\
\langle \textit{expression left } N \rangle &::= \langle \textit{expression postfix } N \rangle \langle \textit{operator left } N \rangle \\
&\quad \langle \textit{expression left } N \rangle \\
&\quad | \quad \langle \textit{expression postfix } N \rangle \\
\langle \textit{expression postfix } N \rangle &::= \langle \textit{expression prefix } N \rangle \langle \textit{operator postfix } N \rangle \\
&\quad | \quad \langle \textit{expression prefix } N \rangle \\
\langle \textit{expression prefix } 20 \rangle &::= \langle \textit{operator prefix } 20 \rangle \langle \textit{application} \rangle \\
&\quad | \quad \langle \textit{application} \rangle \\
\langle \textit{expression prefix } N \rangle &::= \langle \textit{operator prefix } N \rangle \langle \textit{expression none } (N-1) \rangle \\
&\quad | \quad \langle \textit{expression none } (N-1) \rangle \\
\langle \textit{application} \rangle &::= \langle \textit{atomic expression} \rangle \langle \textit{application} \rangle \\
&\quad | \quad \langle \textit{atomic expression} \rangle \\
\langle \textit{atomic expression} \rangle &::= \langle \textit{constant} \rangle \\
&\quad | \quad ( \langle \textit{expression sequence} \rangle ) \\
&\quad | \quad \langle \textit{list expression} \rangle \\
&\quad | \quad \langle \textit{lambda expression} \rangle \\
&\quad | \quad \langle \textit{operator} \rangle \\
&\quad | \quad \langle \textit{identifier} \rangle \\
&\quad | \quad \langle \textit{type name} \rangle \\
\langle \textit{lambda expression} \rangle &::= \{ \mid \langle \textit{formal parameters} \rangle \mid \langle \textit{expression sequence} \rangle \} \\
\langle \textit{list expression} \rangle &::= [ \ ] \\
&\quad | \quad [ \langle \textit{tuple expression} \rangle ] \\
&\quad | \quad [ \langle \textit{tuple expression} \rangle \mid \langle \textit{logical or} \rangle ]
\end{aligned}$$

## 2 Semantics

## 3 Type system