Aurox language specification

Jakub Grobelny

28 maja 2019

1 Syntax

Character classification:

- <*whitespace*> — HT, LF, CR, SPACE
- <*digit*> — 0-9
- <*lowercase*> — underscore or any other lowercase Unicode characters [1]
- <*uppercase*> — any uppercase Unicode characters [2]
- <*special*> — ’-’, ’+’, ’*’, ’/’, ’=’, ’>’, ’<’, ’.’, ’!’, ’@’, ’%’, ’ˆ’, ’ ’, ’&’, ’$’, ’|’

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

⟨*operator*⟩ ::= ⟨*special*⟩
    | ⟨*special*⟩ ⟨*operator*⟩

⟨*keyword*⟩ ::= **let** | **in** | **if** | **and** | **or** | **then**
    | **match** | **else** | **with** | **type** | **import**
    | **define** | **_** | **defop** | **end** | **case**

[1] All characters `X` which satisfy `char_type(X, lower) predicate in SWI Prolog`

[2] All characters `X` which satisfy `char_type(X, upper) predicate in SWI Prolog`

$\langle \mathit{identifier} \rangle ::= \langle \mathit{lowercase} \rangle\ \langle \mathit{alphanum} \rangle$

$\langle \mathit{type\ name} \rangle ::= \langle \mathit{uppercase} \rangle\ \langle \mathit{alphanum} \rangle$

$\langle \mathit{alphanum} \rangle ::= \langle \mathit{alphanum\ char} \rangle\ \langle \mathit{alphanum} \rangle \mid \epsilon$

$\langle \mathit{alphanum\ char} \rangle ::= \langle \mathit{lowercase} \rangle \mid \textbf{?} \mid \langle \mathit{digit} \rangle$

$\langle \mathit{integer} \rangle ::= \langle \mathit{digit} \rangle \mid \langle \mathit{digit} \rangle\ \langle \mathit{integer} \rangle$

$$
\begin{aligned}
\langle \mathit{float} \rangle ::=\ & \langle \mathit{integer} \rangle\ .\ \langle \mathit{digit\ sequence} \rangle\ \langle \mathit{exponent} \rangle\ \textbf{e} \\
\mid\ & \langle \mathit{integer} \rangle\ \langle \mathit{expontent} \rangle
\end{aligned}
$$

$\langle \mathit{digit\ sequence} \rangle ::= \langle \mathit{digit} \rangle \mid \langle \mathit{digit} \rangle\ \langle \mathit{digit\ sequence} \rangle$

$\langle \mathit{e} \rangle \quad ::= \textbf{e} \mid \textbf{E}$

$$
\begin{aligned}
\langle \mathit{exponent} \rangle ::=\ & \langle \mathit{e} \rangle\ \textbf{-}\ \langle \mathit{integer} \rangle \\
\mid\ & \langle \mathit{e} \rangle\ \langle \mathit{integer} \rangle
\end{aligned}
$$

$\langle \mathit{boolean} \rangle ::= \textbf{false} \mid \textbf{true}$

$\langle \mathit{string} \rangle ::= \textbf{"}\ \langle \mathit{char\ sequence} \rangle\ \textbf{"} \mid \textbf{""}$

$\langle \mathit{char} \rangle ::= \textbf{'}\ \langle \mathit{character} \rangle\ \textbf{'}$

$\langle \mathit{char\ sequence} \rangle ::= \langle \mathit{character} \rangle \mid \langle \mathit{character} \rangle\ \langle \mathit{char\ sequence} \rangle$

$\langle \mathit{character} \rangle ::= \text{Unicode} \mid \backslash\backslash \mid \backslash\textbf{b} \mid \backslash\textbf{n} \mid \backslash\textbf{f} \mid \backslash\textbf{a} \mid \backslash\textbf{r} \mid \backslash\textbf{t} \mid \backslash\textbf{0} \mid \backslash\textbf{"} \mid \backslash\textbf{'}$

$$
\begin{aligned}
\langle \mathit{program} \rangle ::=\ & \langle \mathit{operator\ declaration} \rangle\ \langle \mathit{program} \rangle \\
\mid\ & \langle \mathit{import} \rangle\ \langle \mathit{program} \rangle \\
\mid\ & \langle \mathit{expression\ sequence} \rangle\ \langle \mathit{program} \rangle \\
\mid\ & \langle \mathit{definition} \rangle\ \langle \mathit{program} \rangle \\
\mid\ & \epsilon
\end{aligned}
$$

$\langle \mathit{operator\ declaration} \rangle ::= \textbf{defop}\ \langle \mathit{operator} \rangle\ \langle \mathit{integer} \rangle\ \langle \mathit{associativity} \rangle$

$\langle \mathit{associativity} \rangle ::= \textbf{left} \mid \textbf{right} \mid \textbf{none} \mid \textbf{prefix} \mid \textbf{postfix}$

$\langle \mathit{import} \rangle ::= \textbf{import}\ \langle \mathit{import\ list} \rangle\ \textbf{end}$

$\langle\textit{import list}\rangle ::= \epsilon \mid \langle\textit{string}\rangle\ \langle\textit{import list}\rangle$
$\quad\mid\ \langle\textit{type name}\rangle\ \langle\textit{import list}\rangle$

$\langle\textit{definition}\rangle ::= \textbf{define}\ \langle\textit{function name}\rangle\ \langle\textit{formal parameters}\rangle\ \textbf{:}$
$\quad\langle\textit{type}\rangle = \langle\textit{expression sequence}\rangle\ \textbf{end}$

$\langle\textit{function name}\rangle ::= \langle\textit{identifier}\rangle \mid \textbf{(}\ \langle\textit{operator}\rangle\ \textbf{)}$

$\langle\textit{formal parameters}\rangle ::= \langle\textit{variable name}\rangle\ \langle\textit{formal parameters}\rangle \mid \epsilon$

$\langle\textit{variable name}\rangle ::= \langle\textit{identifier}\rangle \mid \textbf{\_}$

$\langle\textit{type}\rangle ::= \langle\textit{function type}\rangle$
$\quad\mid\ \langle\textit{function type}\rangle\ \textbf{,}\ \langle\textit{tupe}\rangle$

$\langle\textit{function type}\rangle ::= \langle\textit{algebraic data type}\rangle$
$\quad\mid\ \langle\textit{function type}\rangle\ \textbf{(-}{>}\textbf{)}\ \langle\textit{algebraic data type}\rangle$

$\langle\textit{algebraic data type}\rangle ::= \langle\textit{type name}\rangle\ \langle\textit{atomic type sequence}\rangle$
$\quad\mid\ \langle\textit{atomic type}\rangle$

$\langle\textit{atomic type sequence}\rangle ::= \langle\textit{atomic type}\rangle\ \langle\textit{atomic type sequence}\rangle \mid \epsilon$

$\langle\textit{atomic type}\rangle ::= \langle\textit{identifier}\rangle \mid \langle\textit{type name}\rangle$
$\quad\mid\ \textbf{[}\ \langle\textit{type}\rangle\ \textbf{]} \mid \textbf{(}\ \langle\textit{type}\rangle\ \textbf{)}$

$\langle\textit{type definition}\rangle ::= \textbf{type}\ \langle\textit{type name}\rangle\ \langle\textit{formal parameters}\rangle\ \textbf{with}$
$\quad\langle\textit{type constructors}\rangle\ \textbf{end}$

$\langle\textit{type constructors}\rangle ::= \textbf{case}\ \langle\textit{type name}\rangle\ \langle\textit{atomic type}\rangle$
$\quad\mid\ \textbf{case}\ \langle\textit{type name}\rangle$

$\langle\textit{expression sequence}\rangle ::= \langle\textit{expression}\rangle$
$\quad\mid\ \langle\textit{expression}\rangle\ \textbf{;}\ \langle\textit{expression sequence}\rangle$

$\langle\textit{expression}\rangle ::= \langle\textit{pattern matching}\rangle \mid \langle\textit{let definition}\rangle$
$\quad\mid\ \langle\textit{conditional expression}\rangle \mid \langle\textit{tuple expression}\rangle$

$\langle\textit{let definition}\rangle ::= \textbf{let}\ \langle\textit{variable name}\rangle\ \textbf{:}\ \langle\textit{type}\rangle =$
$\quad\langle\textit{expression sequence}\rangle\ \textbf{in}\ \langle\textit{expression sequence}\rangle\ \textbf{end}$

$\langle \textit{conditional expression} \rangle$ ::= **if** $\langle \textit{expression sequence} \rangle$ **then**
$\langle \textit{expression sequence} \rangle$ **else** $\langle \textit{expression sequence} \rangle$ **end**

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

$\langle \textit{pattern matching cases} \rangle$ ::= $\langle \textit{pattern case} \rangle$ $\langle \textit{pattern matching cases} \rangle$
| ϵ

$\langle \textit{pattern case} \rangle$ ::= **case** $\langle \textit{pattern} \rangle$ => $\langle \textit{expression sequence} \rangle$

$\langle \textit{pattern} \rangle$ ::= $\langle \textit{deconstructor pattern} \rangle$
| $\langle \textit{deconstructor pattern} \rangle$ **,** $\langle \textit{pattern} \rangle$

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

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

$\langle \textit{list pattern} \rangle$ ::= **[** $\langle \textit{pattern} \rangle$ | $\langle \textit{variable name} \rangle$ **]**
| **[** $\langle \textit{pattern} \rangle$ **]**
| **[ ]**

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

$\langle \textit{tuple expression} \rangle$ ::= $\langle \textit{logical or} \rangle$ **,** $\langle \textit{tuple expression} \rangle$
| $\langle \textit{logical or} \rangle$

$\langle \textit{logical or} \rangle$ ::= $\langle \textit{logical and} \rangle$ **and** $\langle \textit{logical or} \rangle$
| $\langle \textit{logical and} \rangle$

$\langle \textit{logical and} \rangle$ ::= $\langle \textit{expression none 0} \rangle$ **and** $\langle \textit{logical and} \rangle$
| $\langle \textit{expression none 0} \rangle$

⟨*expression none N*⟩ ::= ⟨*expression right N*⟩ ⟨*operator none N*⟩[3]
⟨*expression none N*⟩
| ⟨*expression right N*⟩

⟨*expression right N*⟩ ::= ⟨*expression left N*⟩ ⟨*operator right N*⟩[4]
⟨*expression right N*⟩
| ⟨*expression left N*⟩

⟨*expression left N*⟩ ::= ⟨*expression postfix N*⟩ ⟨*operator left N*⟩[5]
⟨*expression left N*⟩
| ⟨*expression postfix N*⟩

⟨*expression postfix N*⟩ ::= ⟨*expression prefix N*⟩ ⟨*operator postfix N*⟩[6]
| ⟨*expression prefix N*⟩

⟨*expression prefix 20*⟩ ::= ⟨*operator prefix 20*⟩[7] ⟨*application*⟩
| ⟨*application*⟩

⟨*expression prefix N*⟩ ::= ⟨*operator prefix N*⟩[8] ⟨*expression none (N+1)*⟩
| ⟨*expression none (N+1)*⟩

⟨*application*⟩ ::= ⟨*atomic expression*⟩ ⟨*application*⟩
| ⟨*atomic expression*⟩

⟨*atomic expression*⟩ ::= ⟨*constant*⟩
| **(** ⟨*expression sequence*⟩ **)**
| ⟨*list expression*⟩
| ⟨*lambda expression*⟩
| **`** ⟨*operator*⟩
| ⟨*identifier*⟩
| ⟨*type name*⟩

[3] Operator defined with `none` associativity and `N` priority
[4] Operator defined with `right` associativity and `N` priority
[5] Operator defined with `left` associativity and `N` priority
[6] Operator defined with `postfix` associativity and `N` priority
[7] Operator defined with `prefix` associativity and `20` priority
[8] Operator defined with `prefix` associativity and `N` priority

$\langle \textit{lambda expression} \rangle$ ::= **{** | $\langle \textit{formal parameters} \rangle$ | $\langle \textit{expression sequence} \rangle$ **}**

$\langle \textit{list expression} \rangle$ ::= **[ ]**
| **[** $\langle \textit{tuple expression} \rangle$ **]**
| **[** $\langle \textit{tuple expression} \rangle$ | $\langle \textit{logical or} \rangle$ **]**

2 Semantics

$$\frac{\rho(x) = v}{\rho \vdash x \Downarrow v}$$

If c is a constant, then $\overline{c \Downarrow c}$

$$\frac{\rho \vdash e_1 \Downarrow true \quad \rho \vdash e_2 \Downarrow v}{\rho \vdash if\ e_1\ then\ e_2\ else\ e_3\ end \Downarrow v}$$

$$\frac{\rho \vdash e_1 \Downarrow false \quad \rho \vdash e_3 \Downarrow v}{\rho \vdash if\ e_1\ then\ e_2\ else\ e_3\ end \Downarrow v}$$

$$\frac{\rho \vdash e_1 \Downarrow true \quad \rho \vdash e_2 \Downarrow true}{\rho \vdash e_1\ and\ e_2 \Downarrow true}$$

$$\frac{\rho \vdash e_1 \Downarrow false}{\rho \vdash e_1\ and\ e_2 \Downarrow false}$$

$$\frac{\rho \vdash e_1 \Downarrow false \quad \rho \vdash e_2 \Downarrow false}{\rho \vdash e_1\ and\ e_2 \Downarrow false}$$

$$\frac{\rho \vdash e_1 \Downarrow false \quad \rho \vdash e_2 \Downarrow false}{\rho \vdash e_1\ or\ e_2 \Downarrow false}$$

$$\frac{\rho \vdash e_1 \Downarrow true}{\rho \vdash e_1 \ or \ e_2 \Downarrow true}$$

$$\frac{\rho \vdash e_1 \Downarrow true \quad \rho \vdash e_2 \Downarrow true}{\rho \vdash e_1 \ or \ e_2 \Downarrow true}$$

$$\frac{\rho \vdash e_2 \Downarrow v_0 \quad \rho[x \mapsto v_0] \vdash e_1 \Downarrow v}{\rho \vdash \{|x| \ e_1\} \, e_2 \Downarrow v}$$

$$\frac{\rho \vdash e_2 \Downarrow v_0 \quad \rho[x \mapsto v_0] \vdash e_1 \Downarrow v}{\rho \vdash \{|x| \ e_1\} \, e_2 \Downarrow v}$$

$$\frac{}{\rho \vdash [\,] :: [\,]}$$

$$\frac{\rho \vdash e_1 \Downarrow v_1 \quad \rho \vdash [e_2, \, ... \, , \, e_n] \Downarrow [v_2, \, ... \, v_n]}{\rho \vdash [e_1, e_2, \, ... \, , \, e_n] \Downarrow [v_1, v_2, \, ... \, , \, v_n]}$$

$$\frac{\rho \vdash e_1 \Downarrow v_1 \quad \rho \vdash [e_2, \, ... \, , \, e_n] \Downarrow [v_2, \, ... \, v_n] \quad \rho \vdash t \Downarrow [v_{n+1}, \, ... \, v_{n+k}]}{\rho \vdash [e_1, e_2, \, ... \, , \, e_n \, | \, t] \Downarrow [v_1, v_2, \, ... \, , \, v_n, \, v_{n+1} \, ... \, v_{n+k}]}$$

$$\frac{\rho \vdash e_1 \Downarrow v_0 \quad \rho[x \mapsto v_0] \vdash e_2 \Downarrow v}{\rho \vdash let \, x \, := \, e_1 \, in \, e_2 \, end \Downarrow v}$$

$$\frac{\rho \vdash e_1 \Downarrow v_0 \quad \rho \vdash e_2; \, ...; \, e_n \Downarrow v}{\rho \vdash e_1; \, e_2; \, ...; \, e_n \Downarrow v}$$

ρ' means the environment after unification with the pattern.

$$\frac{\rho \vdash e \Downarrow v_0 \quad v_0 = p \quad \rho' \vdash e_1 \Downarrow v}{\rho \vdash match \, e \, with \, case \, p \, \Rightarrow \, e_1 \, end \Downarrow v}$$

$$\frac{\rho \vdash e \Downarrow v_0 \quad v_0 = p_1 \quad \rho' \vdash e_1 \Downarrow v}{\rho \vdash match \, e \, with \, case \, p_1 \, \Rightarrow \, e_1 \, ... \, case \, p_n \, \Rightarrow \, e_n \, end \Downarrow v}$$

$$\frac{\rho \vdash e \Downarrow v_0 \quad v_0 \neq p_1 \quad \rho \vdash match \, e \, with \, case \, p_2 \Rightarrow \, e_2 \, ... \, case \, p_n \Rightarrow \, e_n \Downarrow v}{\rho \vdash match \, e \, with \, case \, p_1 \Rightarrow \, e_1 \, case \, p_2 \Rightarrow e_2 \, ... \, case \, p_n \Rightarrow e_n \, end \Downarrow v}$$

Rules for entire programs. ϵ stands for empty program.

$$\frac{}{\vdash \epsilon}$$

$$\frac{\rho[x \mapsto x] \vdash e_1 \Downarrow v \quad \rho[x \mapsto v] \vdash p_1 \ldots p_n}{\rho \vdash define\, x\ :=\ e_1\, end\ \ p_1 \ldots p_n}$$

$$\frac{\rho \vdash e \Downarrow v \quad \rho \vdash p_1 \ldots p_n}{\rho \vdash e\, p_1 \ldots p_n}$$

3 Type system

$$\frac{\Gamma \vdash e_1 :: \alpha \to \tau \quad \Gamma \vdash e_2 :: \alpha}{\Gamma \vdash e_1\, e_2 :: \tau}$$

$$\frac{\Gamma \vdash c :: Bool \quad \Gamma \vdash e_1 :: \tau \quad \Gamma \vdash e_2 :: \tau}{\Gamma \vdash if\ c\ then\, e_1\ else\, e_2\ end :: \tau}$$

$$\frac{}{\Gamma \vdash [\,] :: [\tau]}$$

$$\frac{\Gamma \vdash e_1 :: \tau \quad \Gamma \vdash [e_2, ..., e_n] :: [\tau]}{\Gamma \vdash [e_1, e_2, ...e_n] :: [\tau]}$$

$$\frac{\Gamma \vdash [e_1, e_2, ...e_n] :: [\tau] \quad \Gamma \vdash e_{n+1} :: [\tau]}{\Gamma \vdash [e_1, e_2, ...e_n \,|\, e_{n+1}] :: [\tau]}$$

$$\frac{\Gamma \vdash e_1 :: \tau_1 \quad \Gamma \vdash e_2 :: \tau_2}{\Gamma \vdash e_1, e_2 :: \tau_1, \tau_2}$$

$$\frac{\Gamma \vdash e_n :: \tau}{\Gamma \vdash e_1; e_2; ...e_n :: \tau}$$

$$\frac{\Gamma \vdash e_1 :: Bool \quad \Gamma \vdash e_2 :: Bool}{\Gamma \vdash e_1\, and\, e_2 :: Bool}$$

$$\frac{\Gamma \vdash e_1 :: Bool \quad \Gamma \vdash e_2 :: Bool}{\Gamma \vdash e_1 \, or \, e_2 :: Bool}$$

$$\frac{\Gamma(x) = \tau}{\Gamma \vdash x :: \tau}$$

$\frac{}{\Gamma \vdash n :: Int}$, where n is an integer

$\frac{}{\Gamma \vdash x :: Float}$, where x is a real number

$$\frac{}{\Gamma \vdash (\,) :: Unit}$$

$$\frac{}{\Gamma \vdash true :: Bool}$$

$$\frac{}{\Gamma \vdash false :: Bool}$$

$\frac{}{\Gamma \vdash s :: String}$, where s is a string.

The following equivalence is true $String \equiv [Char]$

$\frac{}{\Gamma \vdash c :: Char}$, where c is a character

If $\otimes$ is a binary operator, then

$$\frac{\Gamma \vdash \otimes :: \alpha \to \beta \to \tau \quad \Gamma \vdash e_1 :: \alpha \quad \Gamma \vdash e_2 :: \beta}{\Gamma \vdash e_1 \otimes e_2 :: \tau}$$

If $\otimes$ is prefix unary operator, then

$$\frac{\Gamma \vdash \otimes :: \alpha \to \tau \quad \Gamma \vdash e :: \alpha}{\Gamma \vdash \otimes e :: \tau}$$

If $\otimes$ is postfix unary operator, then

$$\frac{\Gamma \vdash \otimes :: \alpha \to \tau \quad \Gamma \vdash e :: \alpha}{\Gamma \vdash e \otimes :: \tau}$$

$$\frac{\Gamma \vdash e_1 :: \alpha \quad \Gamma[x \mapsto \alpha] \vdash e_2 :: \tau}{\Gamma \vdash let\, x \, := \, e_1 \, in\, e_2 \, end :: \tau}$$

$$\frac{\Gamma[x \mapsto \alpha] \vdash e :: \beta}{\Gamma \vdash \{|x| \; e\} :: \alpha \to \beta}$$

$$\frac{\Gamma[x_1 \mapsto \alpha] \vdash \{|x_2 \dots x_n| \; e\} :: \beta}{\Gamma \vdash \{|x_1 \, x_2 \dots x_n| \; e\} :: \alpha \to \beta}$$

$$\frac{\Gamma \vdash e :: Void}{\Gamma \vdash match\, e\, with\, end}$$

$$\frac{\Gamma \vdash e :: \alpha \quad \Gamma \vdash p :: \alpha \quad \Gamma \vdash e_o :: \tau}{\Gamma \vdash match\, e\, with\, case\, p \Rightarrow e_0 :: \tau}$$

$$\frac{\Gamma \vdash p_1 :: \alpha \quad \Gamma \vdash e :: \alpha \quad \Gamma \vdash e_1 :: \tau \quad match\, e\, with\, case\, p_2 \Rightarrow e_2 \, \dots case\, p_n \Rightarrow e_n :: \tau}{\Gamma \vdash match\, e\, with\, case\, p_1 \Rightarrow e_1 \;\; case\, p_2 \Rightarrow e_2 \; \dots case\, p_n \Rightarrow e_n :: \tau}$$

Rules for entire programs. ϵ stands for an empty program.

$$\frac{}{\vdash \epsilon}$$

$$\frac{\Gamma[x \mapsto \tau] \vdash e :: \tau \quad \Gamma[x \mapsto \tau] \vdash p_1 \dots p_n}{\Gamma \vdash define \;\, x \; := \; e \;\, end \;\, p_1 \dots p_n}$$

$$\frac{\Gamma[f \mapsto \tau] \vdash \{|x_1 \, x_2 \dots x_n| \; e\} :: \tau \quad \Gamma[f \mapsto \tau] \vdash \; p_1 \dots p_n}{\Gamma \vdash define \;\, f \, x_1 \, x_2 \dots x_n := \; e \;\, end \;\, p_1 \dots p_n}$$

$$\frac{\Gamma \vdash e :: \tau \quad \Gamma \vdash p_1 \dots p_n}{\Gamma \vdash e \;\; p_1 \dots p_n}$$