COBALT – Comprehension-supported Boolean and Arithmetic Expression with Lists and Tuples

1 INTRODUCTION

COBALT is a toy language for the COSE212 course at Korea University. COBALT stands for **CO**mprehension-supported **B**oolean and **A**rithmetic expressions with **L**ists and **T**uples, and it supports the following features:

- **unit value** (`()`)
- **number values** (`0`, `1`, `-1`, `2`, `-2`, `3`, `-3`, ...)
- **boolean values** (`true` and `false`)
- **arithmetic operators**: negation (`-`), addition (`+`), subtraction (`-`), multiplication (`*`), division (`/`), and modulo (`%`)
- **logical operators**: conjunction (`&&`), disjunction (`||`), and negation (`!`)
- **comparison operators**: equality (`==` and `!=`) and relational (`<`, `>`, `<=`, and `>=`)
- **conditionals** (`if`-`else`)
- **lists** (`Nil`, `::`, and `List`)
- **list functions** (`head`, `tail`, `isEmpty`, `length`, `map`, `flatMap`, and `filter`)
- **list comprehension** (`for`-`yield`)
- **tuples** (`(e1, ..., en)` where $n \geq 2$)
- **tuple projections** (`_1`, `_2`, ...)
- **variable definitions** (`val`)
- **first-class functions** (`=>`) and **function applications**
- **mutually recursive functions** (`def`)

This document is the specification of COBALT. First, Section 2 describes the concrete syntax, and Section 3 describes the abstract syntax with the desugaring rules. Then, Section 4 describes the big-step operational (natural) semantics of COBALT.

2 CONCRETE SYNTAX

The concrete syntax of COBALT is written in a variant of the extended Backus–Naur form (EBNF). The notation `<nt>` denotes a nonterminal, and `"t"` denotes a terminal. We use `?` to denote an optional element and `+` (or `*`) to denote one or more (or zero or more) repetitions of the preceding element. We use `butnot` to denote a set difference to exclude some strings from a producible set of strings. We omit some obvious terminals using the ellipsis (...) notation.

```
// basic elements
<digit>    ::= "0" | "1" | "2" | ... | "9"
<nonzero>  ::= "1" | "2" | ... | "9"
<number>   ::= "-"? <digit>+
<index>    ::= "_" <nonzero> <digit>*
<alphabet> ::= "A" | "B" | "C" | ... | "Z" | "a" | "b" | "c" | ... | "z"
<idstart>  ::= <alphabet> | "_"
<idcont>   ::= <alphabet> | "_" | <digit>
<keyword>  ::= "List" | "Nil" | "def" | "else" | "false" | "for"
             | "if" | "true" | "val" | "yield"
<id>       ::= <idstart> <idcont>* butnot <keyword>
```

```
// expressions
<expr> ::= "()" | <number> | "true" | "false" | <id>
         // unary and binary operators
         | <uop> <expr> | <expr> <bop> <expr>
         // parentheses
         | "(" <expr> ")" | "{" <expr> "}"
         // conditionals
         | "if" "(" <expr> ")" <expr> "else" <expr>
         // lists
         | "Nil" | <expr> "::" <expr> | "List(" <expr> ")"
         // list functions
         | <expr> "." "head" | <expr> "." "tail" | <expr> "." "isEmpty"
         | <expr> "." "length" | <expr> "." "flatMap" "(" <expr> ")"
         | <expr> "." "map" "(" <expr> ")" | <expr> "." "filter" "(" <expr> ")"
         // list comprehension
         | "for" "{" <comp>+ "}" "yield" <expr>
         // tuples
         | "(" <expr> "," <expr> ["," <expr>]* ")"
         // tuple projections
         | <expr> "." <index>
         // first-class functions
         | "(" ")" "=>" <expr> | <id> "=>" <expr>
         | "(" <id> ["," <id>]* ")" "=>" <expr>
         // mutually recursive functions
         | <fdef>+ ";" <expr>
         // function applications
         | <expr> "(" <expr> ")"
// operators
<uop>  ::= "-" | "!"
<bop>  ::= "+" | "-" | "*" | "/" | "%" | "&&" | "||"
         | "==" | "!=" | "<" | "<=" | ">" | ">="
// comprehension elements
<comp> ::= <id> "<-" <expr> ";" ["if" <expr> ";"]*
// function definitions
<fdef> ::= "def" <id> "(" ")" "=" <expr> ";"
         | "def" <id> "(" <id> ["," <id>]* ")" "=" <expr> ";"
```

The precedence and associativity of operators are defined as follows:

Description	Operator	Precedence	Associativity
Unary	-, !	1	right
Multiplicative	*, /, %	2	left
Additive	+, -	3	
List Construction	::	4	right
Relational	<, <=, >, >=	5	left
Equality	==, !=	6	
Logical Conjunction	&&	7	
Logical Disjunction	||	8	

3 ABSTRACT SYNTAX

The abstract syntax of COBALT is defined as follows:

$$
\begin{array}{llll|ll}
\text{Expressions} & \mathbb{E} \ni e ::= () & (\texttt{EUnit}) & & \mid \texttt{Nil} & (\texttt{ENil}) \\
& \mid n & (\texttt{ENum}) & & \mid e :: e & (\texttt{ECons}) \\
& \mid b & (\texttt{EBool}) & & \mid e\texttt{.head} & (\texttt{EHead}) \\
& \mid x & (\texttt{EId}) & & \mid e\texttt{.tail} & (\texttt{ETail}) \\
& \mid e + e & (\texttt{EAdd}) & & \mid e\texttt{.length} & (\texttt{ELength}) \\
& \mid e \times e & (\texttt{EMul}) & & \mid e\texttt{.map}(e) & (\texttt{EMap}) \\
& \mid e \div e & (\texttt{EDiv}) & & \mid e\texttt{.flatMap}(e) & (\texttt{EFlatMap}) \\
& \mid e \ \mathsf{mod}\ e & (\texttt{EMod}) & & \mid e\texttt{.filter}(e) & (\texttt{EFilter}) \\
& \mid e = e & (\texttt{EEq}) & & \mid (e, \ldots, e) & (\texttt{ETuple})\ (\text{length} \geq 2) \\
& \mid e < e & (\texttt{ELt}) & & \mid e. & (\texttt{EProj}) \\
& \mid \texttt{if}\ (e)\ e\ \texttt{else}\ e & (\texttt{EIf}) & & \mid \texttt{val}\ x\texttt{=}e\texttt{;}\ e & (\texttt{EVal}) \\
& & & & \mid \lambda(x, \ldots, x).e & (\texttt{EFun}) \\
& & & & \mid f \ldots f e & (\texttt{ERec}) \\
& & & & \mid e(e, \ldots, e) & (\texttt{EApp})
\end{array}
$$

where

$$
\begin{array}{llllll}
\text{Identifier} & x \in \mathbb{X} & (\texttt{String}) & \text{Boolean} & \mathbb{B} \ni b ::= \texttt{true} \mid \texttt{false} & (\texttt{Boolean}) \\
\text{Index} & i \in \mathbb{Z}^{+} & (\texttt{Int}) & \text{Function Definition} & \mathbb{F} \ni f ::= \texttt{def}\ x(x, \ldots, x)\texttt{=}e\texttt{;} & (\texttt{FunDef}) \\
\text{Number} & n \in \mathbb{Z} & (\texttt{BigInt}) & & &
\end{array}
$$

The semantics of the remaining cases are defined with the following desugaring rules:

$$
\begin{array}{ll}
\mathcal{D}[\![\texttt{-}\ e]\!] & = \mathcal{D}[\![e]\!]\ \texttt{*}\ \texttt{(-1)} \\
\mathcal{D}[\![\texttt{!}\ e]\!] & = \texttt{if}\ \texttt{(}\mathcal{D}[\![e]\!]\texttt{)}\ \texttt{false}\ \texttt{else}\ \texttt{true} \\
\mathcal{D}[\![e_1\ \texttt{-}\ e_2]\!] & = \mathcal{D}[\![e_1]\!]\ \texttt{+}\ \mathcal{D}[\![\texttt{-}\ e_2]\!] \\
\mathcal{D}[\![e_1\ \texttt{\&\&}\ e_2]\!] & = \texttt{if}\ \texttt{(}\mathcal{D}[\![e_1]\!]\texttt{)}\ \mathcal{D}[\![e_2]\!]\ \texttt{else}\ \texttt{false} \\
\mathcal{D}[\![e_1\ \texttt{||}\ e_2]\!] & = \texttt{if}\ \texttt{(}\mathcal{D}[\![e_1]\!]\texttt{)}\ \texttt{true}\ \texttt{else}\ \mathcal{D}[\![e_2]\!] \\
\mathcal{D}[\![e_1\ \texttt{!=}\ e_2]\!] & = \mathcal{D}[\![\texttt{!}\ \texttt{(}e_1\ \texttt{==}\ e_2\texttt{)}]\!] \\
\mathcal{D}[\![e_1\ \texttt{<=}\ e_2]\!] & = \mathcal{D}[\![\texttt{(}e_1\ \texttt{<}\ e_2\texttt{)}\ \texttt{||}\ \texttt{(}e_1\ \texttt{==}\ e_2\texttt{)}]\!] \\
\mathcal{D}[\![e_1\ \texttt{>}\ e_2]\!] & = \mathcal{D}[\![\texttt{!}\ \texttt{(}e_1\ \texttt{<=}\ e_2\texttt{)}]\!] \\
\mathcal{D}[\![e_1\ \texttt{>=}\ e_2]\!] & = \mathcal{D}[\![\texttt{!}\ \texttt{(}e_1\ \texttt{<}\ e_2\texttt{)}]\!] \\
\mathcal{D}[\![\texttt{List(}e_1, \ldots, e_n\texttt{)}]\!] & = \mathcal{D}[\![e_1]\!]\ \texttt{::}\ \ldots\ \texttt{::}\ \mathcal{D}[\![e_n]\!]\ \texttt{::}\ \texttt{Nil} \\
\mathcal{D}[\![e\texttt{.isEmpty}]\!] & = \mathcal{D}[\![e]\!]\ \texttt{==}\ \texttt{Nil}
\end{array}
$$

$$
\mathcal{D}\left[\!\!\left[
\begin{array}{l}
\texttt{for \{} \\
\quad x_1\ \texttt{<-}\ e_1\texttt{;}\texttt{if}\ e_{1,1}\texttt{;}\ \texttt{if}\ e_{1,1}\texttt{;}\ \ldots\texttt{if}\ e_{1,k_1}\texttt{;} \\
\quad \ldots \\
\quad x_n\ \texttt{<-}\ e_n\texttt{;}\texttt{if}\ e_{n,1}\texttt{;}\ \texttt{if}\ e_{n,1}\texttt{;}\ \ldots\texttt{if}\ e_{n,k_n}\texttt{;} \\
\texttt{\} yield}\ e
\end{array}
\right]\!\!\right] =
\begin{array}{l}
\mathcal{D}[\![e_1]\!] \\
\quad \texttt{.filter(}x_1\ \texttt{=>}\ \mathcal{D}[\![e_{1,1}]\!]\texttt{)} \\
\quad \ldots \\
\quad \texttt{.filter(}x_1\ \texttt{=>}\ \mathcal{D}[\![e_{1,k_1}]\!]\texttt{)} \\
\quad \texttt{.flatMap(}x_1\ \texttt{=> \{} \\
\qquad \ldots \\
\qquad \mathcal{D}[\![e_n]\!] \\
\qquad\quad \texttt{.filter(}x_n\ \texttt{=>}\ \mathcal{D}[\![e_{n,1}]\!]\texttt{)} \\
\qquad\quad \ldots \\
\qquad\quad \texttt{.filter(}x_n\ \texttt{=>}\ \mathcal{D}[\![e_{n,k_n}]\!]\texttt{)} \\
\qquad\quad \texttt{.map(}x_n\ \texttt{=>}\ \mathcal{D}[\![e]\!]\texttt{)} \\
\quad \texttt{\})}
\end{array}
$$

The omitted cases recursively apply the desugaring rule to sub-expressions.

4 SEMANTICS

We use the following notations in the semantics:

$$
\begin{array}{llllll}
\text{Value} & \mathbb{V} \ni v ::= () & (\texttt{UnitV}) & \mid (v, \ldots, v) & (\texttt{TupleV}) \\
& \quad \mid n & (\texttt{NumV}) & \mid \langle \lambda(x, \ldots, x).e, \sigma \rangle & (\texttt{CloV}) \\
& \quad \mid b & (\texttt{BoolV}) & & \\
& \quad \mid \texttt{Nil} & (\texttt{NilV}) & & \\
& \quad \mid v :: v & (\texttt{ConsV}) & &
\end{array}
$$

$$
\text{Environment} \qquad \sigma \in \mathbb{X} \xrightarrow{\text{fin}} \mathbb{V} \quad (\texttt{Env})
$$

The big-step operational (natural) semantics of COBALT is defined as follows:

$$\boxed{\sigma \vdash e \Rightarrow v}$$

$$
\texttt{Unit}\ \frac{}{\sigma \vdash () \Rightarrow ()} \qquad
\texttt{Num}\ \frac{}{\sigma \vdash n \Rightarrow n} \qquad
\texttt{Bool}\ \frac{}{\sigma \vdash b \Rightarrow b} \qquad
\texttt{Id}\ \frac{x \in \text{Domain}(\sigma)}{\sigma \vdash x \Rightarrow \sigma(x)}
$$

$$
\texttt{Add}\ \frac{\sigma \vdash e_1 \Rightarrow n_1 \qquad \sigma \vdash e_2 \Rightarrow n_2}{\sigma \vdash e_1 + e_2 \Rightarrow n_1 + n_2} \qquad
\texttt{Mul}\ \frac{\sigma \vdash e_1 \Rightarrow n_1 \qquad \sigma \vdash e_2 \Rightarrow n_2}{\sigma \vdash e_1 \times e_2 \Rightarrow n_1 \times n_2}
$$

$$
\texttt{Div}\ \frac{\sigma \vdash e_1 \Rightarrow n_1 \qquad \sigma \vdash e_2 \Rightarrow n_2 \qquad n_2 \neq 0}{\sigma \vdash e_1 \div e_2 \Rightarrow n_1 \div n_2} \qquad
\texttt{Mod}\ \frac{\sigma \vdash e_1 \Rightarrow n_1 \qquad \sigma \vdash e_2 \Rightarrow n_2 \qquad n_2 \neq 0}{\sigma \vdash e_1\ \texttt{mod}\ e_2 \Rightarrow n_1\ \texttt{mod}\ n_2}
$$

$$
\texttt{Eq}\ \frac{\sigma \vdash e_1 \Rightarrow v_1 \qquad \sigma \vdash e_2 \Rightarrow v_2}{\sigma \vdash e_1 = e_2 \Rightarrow \texttt{eq}(v_1, v_2)} \qquad
\texttt{Lt}\ \frac{\sigma \vdash e_1 \Rightarrow n_1 \qquad \sigma \vdash e_2 \Rightarrow n_2}{\sigma \vdash e_1 < e_2 \Rightarrow n_1 < n_2}
$$

$$
\texttt{If}_T\ \frac{\sigma \vdash e_0 \Rightarrow \texttt{true} \qquad \sigma \vdash e_1 \Rightarrow v_1}{\sigma \vdash \texttt{if}\ (e_0)\ e_1\ \texttt{else}\ e_2 \Rightarrow v_1} \qquad
\texttt{If}_F\ \frac{\sigma \vdash e_0 \Rightarrow \texttt{false} \qquad \sigma \vdash e_2 \Rightarrow v_2}{\sigma \vdash \texttt{if}\ (e_0)\ e_1\ \texttt{else}\ e_2 \Rightarrow v_2}
$$

$$
\texttt{Nil}\ \frac{}{\sigma \vdash \texttt{Nil} \Rightarrow \texttt{Nil}} \qquad
\texttt{Cons}\ \frac{\sigma \vdash e_1 \Rightarrow v_1 \qquad \sigma \vdash e_2 \Rightarrow v_2 \qquad (v_2 = \texttt{Nil} \lor v_2 = \_ :: \_)}{\sigma \vdash e_1 :: e_2 \Rightarrow v_1 :: v_2}
$$

$$
\texttt{Head}\ \frac{\sigma \vdash e \Rightarrow v_1 :: v_2}{\sigma \vdash e\texttt{.head} \Rightarrow v_1} \qquad
\texttt{Tail}\ \frac{\sigma \vdash e \Rightarrow v_1 :: v_2}{\sigma \vdash e\texttt{.tail} \Rightarrow v_2} \qquad
\texttt{Length}\ \frac{\sigma \vdash e \Rightarrow v \qquad \texttt{length}(v) = n}{\sigma \vdash e\texttt{.length} \Rightarrow n}
$$

$$
\texttt{Map}\ \frac{\sigma \vdash e_1 \Rightarrow v_1 \qquad \sigma \vdash e_2 \Rightarrow v_2 \qquad \texttt{map}(v_1, v_2) = v'}{\sigma \vdash e_1\texttt{.map}(e_2) \Rightarrow v'}
$$

$$
\texttt{FlatMap}\ \frac{\sigma \vdash e_1 \Rightarrow v_1 \qquad \sigma \vdash e_2 \Rightarrow v_2 \qquad \texttt{join}(\texttt{map}(v_1, v_2)) = v'}{\sigma \vdash e_1\texttt{.flatMap}(e_2) \Rightarrow v'}
$$

$$
\texttt{Filter}\ \frac{\sigma \vdash e_1 \Rightarrow v_1 \qquad \sigma \vdash e_2 \Rightarrow v_2 \qquad \texttt{filter}(v_1, v_2) = v'}{\sigma \vdash e_1\texttt{.filter}(e_2) \Rightarrow v'}
$$

$$\texttt{Tuple}\ \frac{\sigma \vdash e_1 \Rightarrow v_1 \qquad \dots \qquad \sigma \vdash e_n \Rightarrow v_n}{\sigma \vdash (e_1, \dots, e_n) \Rightarrow (v_1, \dots, v_n)} \qquad \texttt{Proj}\ \frac{\sigma \vdash e \Rightarrow (v_1, \dots, v_n)}{\sigma \vdash e.i \Rightarrow v_i}$$

$$\texttt{Val}\ \frac{\sigma \vdash e_1 \Rightarrow v_1 \qquad \sigma[x \mapsto v_1] \vdash e_2 \Rightarrow v_2}{\sigma \vdash \texttt{val}\ x\texttt{=}e_1\texttt{;}\ e_2 \Rightarrow v_2} \qquad \texttt{Fun}\ \frac{}{\sigma \vdash \lambda(x_1, \dots, x_n).e \Rightarrow \langle \lambda(x_1, \dots, x_n).e, \sigma \rangle}$$

$$\texttt{Rec}\ \frac{\sigma' = \sigma[x_1 \mapsto \langle \lambda(x_{1,1}, \dots, x_{1,k_1}).e_1, \sigma' \rangle, \dots x_n \mapsto \langle \lambda(x_{n,1}, \dots, x_{n,k_n}).e_n, \sigma' \rangle,] \qquad \sigma' \vdash e \Rightarrow v}{\sigma \vdash \texttt{def}\ x_1(x_{1,1}, \dots, x_{1,k_1})\texttt{=}e_1\texttt{;} \dots \texttt{def}\ x_n(x_{n,1}, \dots, x_{n,k_n})\texttt{=}e_n\texttt{;}\, e \Rightarrow v}$$

$$\texttt{App}\ \frac{\sigma \vdash e_0 \Rightarrow v_0 \qquad \sigma \vdash e_1 \Rightarrow v_1 \qquad \dots \qquad \sigma \vdash e_n \Rightarrow v_n \qquad \mathsf{app}(v_0, [v_1, \dots, v_n]) = v}{\sigma \vdash e_0(e_1, \dots, e_n) \Rightarrow v}$$

with the following auxiliary partial functions, and $X \rightharpoonup Y$ denotes a partial function from X to Y.

$$\boxed{\mathsf{eq} : \mathbb{V} \times \mathbb{V} \rightharpoonup \mathbb{B}}$$

$$\begin{aligned}
\mathsf{eq}((), ()) &= \texttt{true} \\
\mathsf{eq}(n_1, n_2) &= (n_1 = n_2) \\
\mathsf{eq}(b_1, b_2) &= (b_1 = b_2) \\
\mathsf{eq}(\texttt{Nil}, \texttt{Nil}) &= \texttt{true} \\
\mathsf{eq}(\texttt{Nil}, _ :: _) &= \texttt{false} \\
\mathsf{eq}(_ :: _, \texttt{Nil}) &= \texttt{false}
\end{aligned}$$

$$\mathsf{eq}(v_h :: v_t, v'_h :: v'_t) = \begin{cases} \texttt{false} & \text{if } \mathsf{eq}(v_h, v'_h) = \texttt{false} \\ \mathsf{eq}(v_t, v'_t) & \text{otherwise} \end{cases}$$

$$\mathsf{eq}((v_1, \dots, v_n), (v'_1, \dots, v'_m)) = \begin{cases} \texttt{true} & \text{if } n = 0 \wedge m = 0 \\ \texttt{false} & \text{if } n = 0 \wedge m > 0 \\ \texttt{false} & \text{if } n > 0 \wedge m = 0 \\ \texttt{false} & \text{if } \mathsf{eq}(v_1, v'_1) = \texttt{false} \\ b & \text{otherwise} \end{cases}$$

where $b = \mathsf{eq}((v_2, \dots, v_n), (v'_2, \dots, v'_m))$

$$\boxed{\mathsf{length} : \mathbb{V} \rightharpoonup \mathbb{Z}}$$

$$\begin{aligned}
\mathsf{length}(\texttt{Nil}) &= 0 \\
\mathsf{length}(v_h :: v_t) &= 1 + \mathsf{length}(v_t)
\end{aligned}$$

$$\boxed{\mathsf{map} : \mathbb{V} \times \mathbb{V} \rightharpoonup \mathbb{V}}$$

$$\begin{aligned}
\mathsf{map}(\texttt{Nil}, _) &= \texttt{Nil} \\
\mathsf{map}(v_h :: v_t, v_f) &= \mathsf{app}(v_f, [v_h]) :: \mathsf{map}(v_t, v_f)
\end{aligned}$$

$$\boxed{\mathsf{join} : \mathbb{V} \rightharpoonup \mathbb{V}}$$

$$\begin{aligned}
\mathsf{join}(\texttt{Nil}) &= \texttt{Nil} \\
\mathsf{join}(\texttt{Nil} :: v_t) &= \mathsf{join}(v_t) \\
\mathsf{join}((v_h :: v_t) :: v'_t) &= v_h :: \mathsf{join}(v_t :: v'_t)
\end{aligned}$$

$$\boxed{\mathsf{filter} : \mathbb{V} \times \mathbb{V} \rightharpoonup \mathbb{V}}$$

$$\begin{aligned}
\mathsf{filter}(\texttt{Nil}, _) &= \texttt{Nil} \\
\mathsf{filter}(v_h :: v_t, v_f) &= \begin{cases} v_h :: \mathsf{filter}(v_t, v_f) & \text{if } \mathsf{app}(v_f, [v_h]) = \texttt{true} \\ \mathsf{filter}(v_t, v_f) & \text{if } \mathsf{app}(v_f, [v_h]) = \texttt{false} \end{cases}
\end{aligned}$$

$$\boxed{\mathsf{app} : \mathbb{V} \times \mathbb{V}^* \rightharpoonup \mathbb{V}}$$

$$\mathsf{app}(\langle \lambda(x_1, \dots, x_m).e, \sigma \rangle, [v_1, \dots, v_n]) = \begin{cases} v & \text{if } m \le n \wedge \sigma[x_1 \mapsto v_1, \dots, x_m \mapsto v_m] \vdash e \Rightarrow v \\ v & \text{if } m > n \wedge \sigma[x_1 \mapsto v_1, \dots, x_n \mapsto v_n, x_{n+1} \mapsto (), \dots, x_m \mapsto ()] \vdash e \Rightarrow v \end{cases}$$