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:

- **number (integer) 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 operations** (`head`, `tail`, `isEmpty`, `map`, `flatMap`, `filter`), and `foldLeft`
- **list comprehension** (`for-yield`)
- **tuples** (`(e1, ..., en)` where $n \geq 2$)
- **tuple projections** (`_1`, `_2`, . . .)
- **immutable variable definitions** (`val`)
- **first-class functions** (`=>`)
- **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 and list operations
         | "Nil" | <expr> "::" <expr> | "List(" <expr> ")"
         | <expr> "." "head" | <expr> "." "tail"
         | <expr> "." "isEmpty" | <expr> "." "map" "(" <expr> ")"
         | <expr> "." "flatMap" "(" <expr> ")"
         | <expr> "." "filter" "(" <expr> ")"
         | <expr> "." "foldLeft" "(" <expr>, <expr> ")"
         // list comprehensions
         | "for" "{" <comp>+ "}" "yield" <expr>
         // tuples and tuple projections
         | "(" <expr> "," <expr> ["," <expr>]* ")"
         | <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:

Expressions	$\mathbb{E} \ni e ::= n$	(ENum)	$\mid$ `Nil`	(ENil)
	$\mid b$	(EBool)	$\mid e :: e$	(ECons)
	$\mid x$	(EId)	$\mid e$`.head`	(EHead)
	$\mid e + e$	(EAdd)	$\mid e$`.tail`	(ETail)
	$\mid e * e$	(EMul)	$\mid e$`.map`(e)	(EMap)
	$\mid e \;/\; e$	(EDiv)	$\mid e$`.flatMap`(e)	(EFlatMap)
	$\mid e \;\%\; e$	(EMod)	$\mid e$`.filter`(e)	(EFilter)
	$\mid e == e$	(EEq)	$\mid e$`.foldLeft`(e, e)	(EFoldLeft)
	$\mid e < e$	(ELt)	$\mid (e, \ldots, e)$	(ETuple) (length ≥ 2)
	$\mid$ `if` (e) e `else` e	(EIf)	$\mid e.$	(EProj)
			$\mid$ `val` $x = e;\; e$	(EVal)
			$\mid \lambda(x, \ldots, x).e$	(EFun)
			$\mid f \ldots fe$	(ERec)
			$\mid e(e, \ldots, e)$	(EApp)

where

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

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

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

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

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

4 SEMANTICS

We use the following notations in the semantics:

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

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

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

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

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

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

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

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

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

$$\texttt{Head}\ \frac{\sigma \vdash e \Rightarrow v_1 :: v_2}{\sigma \vdash e\texttt{.head} \Rightarrow v_1} \qquad \texttt{Tail}\ \frac{\sigma \vdash e \Rightarrow v_1 :: v_2}{\sigma \vdash e\texttt{.tail} \Rightarrow v_2}$$

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

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

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

$$\texttt{FoldLeft}\ \frac{\sigma \vdash e_1 \Rightarrow v_1 \qquad \sigma \vdash e_2 \Rightarrow v_2 \qquad \sigma \vdash e_3 \Rightarrow v_3 \qquad \text{foldLeft}(v_1, v_2, v_3) = v'}{\sigma \vdash e_1\texttt{.foldLeft}(e_2, e_3) \Rightarrow v'}$$

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

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

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

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

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

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

$$\begin{array}{ll|ll}
\mathsf{eq}(n, n) & = \texttt{true} & \mathsf{eq}(\texttt{Nil}, _ :: _) & = \texttt{false} \\
\mathsf{eq}(b, b) & = \texttt{true} & \mathsf{eq}(_ :: _, \texttt{Nil}) & = \texttt{false} \\
\mathsf{eq}(\texttt{Nil}, \texttt{Nil}) & = \texttt{true} & \mathsf{eq}(v_h :: v_t, v'_h :: v'_t) & = \mathsf{eq}(v_h, v'_h) \wedge \mathsf{eq}(v_t, v'_t)
\end{array}$$

Note that eq is a total function and should return `false` for the cases not listed above.

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

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

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

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

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

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

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

$$\begin{array}{ll}
\mathsf{foldLeft}(\texttt{Nil}, v_i, _) & = v_i \\
\mathsf{foldLeft}(v_h :: v_t, v_i, v_f) & = \mathsf{foldLeft}(v_t, \mathsf{app}(v_f, [v_i, v_h]), v_f)
\end{array}$$

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

$$\mathsf{app}(\langle \lambda(x_1, \dots, x_n).e, \sigma \rangle, [v_1, \dots, v_n]) = v \quad \text{where} \quad \sigma[x_1 \mapsto v_1, \dots, x_n \mapsto v_n] \vdash e \Rightarrow v$$