Module Interface Specification for the Companion Cube Calculator (C^3)

Geneva Smith

December 17, 2017

1 Revision History

Date	Version	Notes
December 17, 2017	1.2	Updated the Control Flow specification to match the resulting implementation
December 7, 2017	1.1.1	Revised the operator data structure with missing "get" operator, a new exception, and seperated the `numOperands` Integer state variable into three Boolean state variables; added terminator variables to Solver module
December 5, 2017	1.1	Added a specification for an operator data structure
November 27, 2017	1.0	Initial draft completed

2 Symbols, Abbreviations and Acronyms

See SRS Documentation at https://github.com/GenevaS/CAS741/tree/master/Doc/SRS for project symbols, abbreviations, and acronyms.

Contents

1 Revision History i

2 Symbols, Abbreviations and Acronyms ii

3 Introduction 1

4 Notation 2

5 Module Decomposition 3

6 MIS of the Control Flow Module 4
6.1 Module . . . 4
6.2 Uses . . . 4
6.3 Syntax . . . 4
6.3.1 Exported Access Programs . . . 4
6.4 Semantics . . . 4
6.4.1 State Variables . . . 4
6.4.2 Access Routine Semantics . . . 4

7 MIS of the User Input Module 7
7.1 Module . . . 7
7.2 Uses . . . 7
7.3 Syntax . . . 7
7.3.1 Exported Access Programs . . . 7
7.4 Semantics . . . 7
7.4.1 Environment Variables . . . 7
7.4.2 State Variables . . . 7
7.4.3 Assumptions . . . 7
7.4.4 Access Routine Semantics . . . 8

8 MIS of the Interval Conversion Module 9
8.1 Module . . . 9
8.2 Uses . . . 9
8.3 Syntax . . . 9
8.3.1 Exported Access Programs . . . 9
8.4 Semantics . . . 9
8.4.1 State Variables . . . 9
8.4.2 Assumptions . . . 9
8.4.3 Access Routine Semantics . . . 9

9 MIS of the Equation Conversion Module **11**
9.1 Module 11
9.2 Uses 11
9.3 Syntax 11
9.3.1 Exported Constants 11
9.3.2 Exported Access Programs 11
9.4 Semantics 11
9.4.1 State Variables 11
9.4.2 Assumptions 11
9.4.3 Access Routine Semantics 12

10 MIS of the Variable Consolidation Module **14**
10.1 Module 14
10.2 Uses 14
10.3 Syntax 14
10.3.1 Exported Access Programs 14
10.4 Semantics 14
10.4.1 State Variables 14
10.4.2 Assumptions 14
10.4.3 Access Routine Semantics 14

11 MIS of the Range Solver Module **16**
11.1 Module 16
11.2 Uses 16
11.3 Syntax 16
11.3.1 Exported Constants 16
11.3.2 Exported Access Programs 16
11.4 Semantics 16
11.4.1 State Variables 16
11.4.2 Assumptions 16
11.4.3 Access Routine Semantics 16

12 MIS of the Output Module **18**
12.1 Module 18
12.2 Uses 18
12.3 Syntax 18
12.3.1 Exported Access Programs 18
12.4 Semantics 18
12.4.1 State Variables 18
12.4.2 Environment Variables 18
12.4.3 Assumptions 18
12.4.4 Access Routine Semantics 18

13 MIS of the Interval Data Structure Module **20**
13.1 Module 20
13.2 Uses 20
13.3 Syntax 20
13.3.1 Exported Access Programs 20
13.4 Semantics 20
13.4.1 State Variables 20
13.4.2 Access Routine Semantics 20

14 MIS of the Equation Data Structure Module **22**
14.1 Module 22
14.2 Uses 22
14.3 Syntax 22
14.3.1 Exported Access Programs 22
14.4 Semantics 22
14.4.1 State Variables 22
14.4.2 Assumptions 22
14.4.3 Access Routine Semantics 23

15 MIS of the Operator Data Structure Module **24**
15.1 Module 24
15.2 Uses 24
15.3 Syntax 24
15.3.1 Exported Access Programs 24
15.4 Semantics 24
15.4.1 State Variables 24
15.4.2 Assumptions 25
15.4.3 Access Routine Semantics 25

16 Appendix **28**

3 Introduction

The following document details the Module Interface Specifications for the Companion Cube Calculator (C^3), a mathematical tool which determines the range of a user-specified function given the domains of the function's variables. The calculations are performed using interval arithmetic.

It is assumed that the chosen implementation language will automatically check that the appropriate number of inputs are provided to a function and that all inputs are of the expected type. Therefore, these exceptions are not listed in this specification.

Complementary documents include the System Requirement Specifications and Module Guide. The full documentation and implementation can be found at:

https://github.com/GenevaS/CAS741

4 Notation

The structure of the MIS for modules comes from Hoffman and Strooper (1995), with the addition that template modules have been adapted from Ghezzi et al. (2003). The mathematical notation comes from Chapter 3 of Hoffman and Strooper (1995). For instance, the symbol := is used for a multiple assignment statement and conditional rules follow the form $(c_1 \Rightarrow r_1 | c_2 \Rightarrow r_2 | ... | c_n \Rightarrow r_n)$.

The following table summarizes the primitive data types used by Companion Cube Calculator.

Data Type	**Notation**	**Description**
Boolean	$\mathbb{B}$	The set of $\{True, False\}$
Integer	$\mathbb{Z}$	Any whole number in (-∞, ∞)
Real	$\mathbb{R}$	Any number in (-∞, ∞)
String	$char^n$	A sequence of alphanumeric and special characters

The specification of Companion Cube Calculator uses some derived data types: sequences and strings. Sequences are lists filled with elements of the same data type. Strings are sequences of characters. In addition, Companion Cube Calculator uses functions, which are defined by the data types of their inputs and outputs. Local functions are described by giving their type signature followed by their specification.

5 Module Decomposition

The following table is taken directly from the Module Guide document for this project. It can be found at https://github.com/GenevaS/CAS741/blob/master/Doc/Design/MG.

Level 1	**Level 2**
Hardware-Hiding Module	-
Behaviour-Hiding Module	Control Flow Module User Input Module Interval Conversion Module Equation Conversion Module Variable Consolidation Module Range Solver Module Output Module
Software Decision Module	Interval Data Structure Module Equation Data Structure Module Operator Data Structure Module

Table 1: Module Hierarchy

6 MIS of the Control Flow Module

The Control Flow module is the only access point that external applications should use when implementing the Companion Cube Calculator. This affords the freedom to create any type of user interface without changing any of the underlying structure. In some cases, this means that a Control Flow access program simply returns the outputs from other module access programs without modifying them.

6.1 Module

ControlFlow

6.2 Uses

Input (Section 7), EquationConversion (Section 9), Consolidate (Section 10), Solver (Section 11), Output (Section 12), IntervalStruct (Section 13), EquationStruct (Section 14)

6.3 Syntax

6.3.1 Exported Access Programs

Name	**In**	**Out**	**Exceptions**
Initialize	-	$Boolean$	-
ControlFile	$String$	$String^n$	-
ControlDirect	$String$, $String$	$String^n$	-
GetSuccessCode	-	Int	-
GetVariableInfo	-	$String^{n \times 3}$	-

6.4 Semantics

6.4.1 State Variables

- $hasRun : Boolean$
- $successCode : Int$

6.4.2 Access Routine Semantics

Initialize():

- output: $out := success$ where $success$ is the output of the ConfigureParser access program from the EquationConversion module
- exception: N/A

ControlFile(fileName):

- output: $out := inputs$ where $inputs$ is the output of the ReadFile access program from the Input module

- exception: N/A

ControlDirect($equationString, variableListString$):

- transition: Updates the $successCode$ state variable with the return value of the ConvertAndCheckInputs access program from the Consolidate module. If the ControlDirect access program completes successfully, update the $hasRun$ state variable to $True$.

- output: $out := results$ where:

  - The program completed successfully:
    $results$ is the sequence $range, equationTree$. The value for $range$ is the output of the PrintInterval access program from the Output module and the value for $equationTree$ is the output of the PrintEquationTree access program from the Output module.
  - The program was not completed successfully, $results$ is $NULL$

```
results = NULL
successCode = Consolidate.ConvertAndCheckInputs(
    equationString,
    variableListString,
    Solver.GetValidOperators(),
    Solver.GetValidTerminators())
if successCode = 0
  range = Solver.FindRange(
      Consolidate.GetEquationStruct(),
      Consolidate.GetIntervalStructList())
  if range != NULL
    results = {Output.PrintInterval(range),
               Output.PrintEquationTree(Consolidate.
                   GetEquationStruct())}
    hasRun = TRUE
return results
```

GetSuccessCode():

- output: $out := successCode$

- exception: N/A

GetVariableInfo():

- output: $out := varInfoList$ where:

```
    Initialize varInfoList = NULL
    if hasRun is TRUE
      intervalList = Consolidate.GetIntervalStructList()
      if the interval list contains data
        foreach(interval in intervalList)
          varInfoList.Add(interval.GetVariableName,
                          interval.GetMinBound,
                          interval.GetMaxBound)
return varInfoList
```

- exception: N/A

7 MIS of the User Input Module

[Should GetUserInputs take a string is input for when the user wants to obtain the inputs from a file. —SS] [Do you really want the output to be a sequence of strings? What about defining an ADT that stores this information, or an abstract object, if there is only one? —SS]

7.1 Module

Input

7.2 Uses

N/A

7.3 Syntax

7.3.1 Exported Access Programs

Name	In	Out	Exceptions
GetUserInputs	-	$String^n$	IN_BAD_FILE, IN_EMPTY_FILE

7.4 Semantics

7.4.1 Environment Variables

$inputFile : String^n$

7.4.2 State Variables

N/A

7.4.3 Assumptions

- The GetInputMethod function accepts user inputs from files or as direct inputs (From SRS R1).
- If the user chooses to enter their values via a file, it must be formatted such that:
  - The user equation on the first line
  - The list of variable names and interval values associated with the user equation; each name/value set is on its own line and is of the form $varName, minBound$, $maxBound$ [What happens if the variables in the equation do not match the list of variable names? —SS]

- The output of the GetUserInputs function is a list of *String* where the first item is the user equation. The remaining items are the variable names and interval values such that every set of three values represents one data set.

7.4.4 Access Routine Semantics

GetUserInputs():

- transition: If the user has chosen to enter their values via a file, $inputFile$ is associated with the provided file name.
- output: $out := eqString || varList$
- exception: $exc :=$
$(\nexists inputFile \lor \neg Read(inputFile) \Rightarrow IN\_BAD\_FILE)$
$|$
$(Read(inputFile) == \emptyset \Rightarrow IN\_EMPTY\_FILE)$

8 MIS of the Interval Conversion Module

[Why even create the intermediate string form? Why not go directly to the intervalStruct type? —SS]

8.1 Module

IntervalConversion

8.2 Uses

IntervalStruct (Section 13)

8.3 Syntax

8.3.1 Exported Access Programs

Name	In	Out	Exceptions
MakeInterval	$String^3$	$intervalStruct$	IVC_EMPTY_VARNAME, IVC_CONV_ERR_MIN, IVC_CONV_ERR_MAX, IVC_NO_BOUNDS, IVC_NO_MIN, IVC_NO_MAX

8.4 Semantics

8.4.1 State Variables

N/A

8.4.2 Assumptions

- Ensuring that $min \leq max$ is handled by the IntervalStruct (Section 13) module.

8.4.3 Access Routine Semantics

MakeInterval($varName, min, max$):

- output: $out := newInterval$ [Where is newInterval defined? —SS]
- exception: $exc :=$
  $(varName = \text{``''} \Rightarrow IVC\_EMPTY\_VARNAME)$
  $|$
  $(ToReal(min) \notin \mathbb{R} \Rightarrow IVC\_CONV\_ERR\_MIN)$

$|$

$(ToReal(max) \notin \mathbb{R} \Rightarrow IVC\_CONV\_ERR\_MAX)$

$|$

$(min = max = \text{“”} \Rightarrow IVC\_NO\_BOUNDS)$

$|$

$(min = \text{“”} \wedge max \neq \text{“”} \Rightarrow IVC\_NO\_MIN)$

$|$

$(min \neq \text{“”} \wedge max = \text{“”} \Rightarrow IVC\_NO\_MAX)$

9 MIS of the Equation Conversion Module

9.1 Module

EquationConversion

9.2 Uses

EquationStruct (Section 14), OperatorStruct (Section 15)

9.3 Syntax

9.3.1 Exported Constants

- $VARTOKEN : String$
- $CONSTTOKEN : String$

9.3.2 Exported Access Programs

Name	In	Out	Exceptions
ConfigureParser	$operatorStruct^n$, $String^{n\text{x}2}$	$Boolean$	EQC_NO_OPS, EQC_INVALID_OP, EQC_UNBALANCED_TERMINATOR
MakeEquationTree	$String$	$equationStruct$	EQC_CONST_FUNC, EQC_IMPLICIT_MULT, EQC_INCOMPLETE_OP, EQC_UNSUPPORTED_OP
GetVariableToken	-	$String$	-
GetConstToken	-	$String$	-
GetVariableList	-	$String^n$	-

9.4 Semantics

9.4.1 State Variables

- $variableList : String^n$
- $variableStringPattern : String$

9.4.2 Assumptions

- The ConfigureParser function will always be called before any other function in this module.

- The MakeEquationTree function will always be called before the GetVariableList function, otherwise it will not contain any data.

9.4.3 Access Routine Semantics

ConfigureParser($operators, terminators$):

- transition: The value of $variableStringPattern$ is updated so that operators and terminators are not matched when the module is searching for variables.
- output: $out := success$
- exception: $exc :=$
  $(operators = \emptyset \Rightarrow EQC\_NO\_OPS)$
  $|$
  $(\exists op \in operators | op.IsUnary == False \wedge op.IsBinary == False \Rightarrow EQC\_INVALID\_OP)$
  $|$
  $(\exists t[i][2] \in terminators | t[i][2] == \text{""} \Rightarrow EQC\_UNBALANCED\_TERMINATOR)$

MakeEquationTree($userEquation$):

- transition: The value of $variableList$ is updated with new variable names as they are encountered during equation processing.
- output: $out := equationTreeRoot$ [Where is this output defined? I would expect it to be defined in terms of the input parameters. —SS]
- exception: $exc :=$
  $(ToReal(userEquation) \in \mathbb{R} \Rightarrow EQC\_CONST\_FUNC)$
  $|$
  $(\exists subEq | subEq = \{subEq_1, subEq_2\} \wedge subEq_1 \in \mathbb{R} \wedge subEq_2 \in variableList$
  $\Rightarrow EQC\_IMPLICIT\_MULT)$
  $|$
  $(\exists op \in userEquation | (NULL < op > userEquation) \vee (userEquation < op > NULL) \Rightarrow EQC\_INCOMPLETE\_OP)$
  $|$
  $(\exists op | op \in userEquation \wedge op \notin supportedOperations \Rightarrow EQC\_UNSUPPORTED\_OP)$

GetVariableToken():

- output: $out := VARTOKEN$
- exception: N/A

GetConstToken():

- output: $out := CONSTTOKEN$

- exception: N/A

GetVariableList():

- output: $out := variableList$
- exception: N/A

10 MIS of the Variable Consolidation Module

10.1 Module

Consolidate

10.2 Uses

IntervalConversion (Section 8), EquationConversion (Section 9), IntervalStruct (Section 13), EquationStruct (Section 14)

10.3 Syntax

10.3.1 Exported Access Programs

Name	In	Out	Exceptions
ConvertAndCheckInputs	$String$, $String^n$, $operatorStruct^n$	-	VC_MISSING_VARS, VC_EXTRA_VARS, VC_NO_FUNCTION, VC_INVALID_VARNAME
GetEquationStruct	-	$equationStruct$	-
GetIntervalStructList	-	$intervalStruct^n$	-

10.4 Semantics

10.4.1 State Variables

- $equationTreeRoot : equationStruct$
- $intervalList : intervalStruct^n$

10.4.2 Assumptions

- The ConvertAndCheckInputs function will change the state variables before the GetEquationStruct or GetIntervalStructList functions are called.

10.4.3 Access Routine Semantics

ConvertAndCheckInputs(eqString, varList, operators):

- transition: The state variables $equationTreeRoot$ and $intervalList$ will be assigned the values that result from a successful parse and consolidation process. [How is this done? If you cannot represent it formally, maybe a pseudo code algorithm is available? —SS]
- output: N/A

- exception: $exc :=$
  $(\exists var | var \in eqString \wedge var \notin varList \Rightarrow VC\_MISSING\_VARS)$
  $|$
  $(\exists var | var \notin eqString \wedge var \in varList \Rightarrow VC\_EXTRA\_VARS)$
  $|$
  $(eqString == \text{“”} \Rightarrow VC\_NO\_FUNCTION)$
  $|$
  $(\exists varName \supset \{+, -, *, \hat{}, (,)\} \Rightarrow VC\_INVALID\_VARNAME)$

GetEquationStruct():

- output: $out := equationTreeRoot$
- exception: N/A

GetIntervalStructList():

- output: $out := intervalList$
- exception: N/A

11 MIS of the Range Solver Module

11.1 Module

Solver

11.2 Uses

IntervalStruct (Section 13), EquationStruct (Section 14), OperatorStruct (Section 15)

11.3 Syntax

11.3.1 Exported Constants

- $supportedOps : operatorStruct^n$
- $supportedTerminators : String^{n\text{x}2}$

11.3.2 Exported Access Programs

Name	In	Out	Exceptions
GetValidOperators	-	$operatorStruct^n$	-
GetValidTerminators	-	$String^{n\text{x}2}$	-
FindRange	$equationStruct$, $intervalStruct^n$	$intervalStruct$	SOL_UNSUPPORTED_OP

11.4 Semantics

11.4.1 State Variables

N/A

11.4.2 Assumptions

- The type of $intervalStruct^n$ accepts `NULL` as a valid value.

11.4.3 Access Routine Semantics

GetValidOperators():

- output: $out := supportedOps$
- exception: N/A

GetValidTerminators():

- output: $out := supportedTerminators$

- exception: N/A

FindRange($eStruct, ivStructList$):

- output: $out := eqRange$
- exception: $exc :=$
  $((\exists op \in eStruct \wedge op \notin supportedOps)$
  $\vee (\exists iv1, iv2 \in ivStructList \wedge \nexists op \in supportedOps | op(iv1, iv2) \vee op(iv2, iv1))$
  $\Rightarrow SOL\_UNSUPPORTED\_OP)$

12 MIS of the Output Module

12.1 Module

Output

12.2 Uses

IntervalStruct (Section 13), EquationStruct (Section 14)

12.3 Syntax

12.3.1 Exported Access Programs

Name	In	Out	Exceptions
PrintInterval	*intervalStruct*	-	-
PrintEquationTree	*equationStruct*	-	-

12.4 Semantics

12.4.1 State Variables

N/A

12.4.2 Environment Variables

- cmd: the command-line interface
- win: a 2D sequence of pixels displayed on the screen

12.4.3 Assumptions

- There are no exceptions in this module because it is assumed that only well-formed inputs will be passed in. This assumption is made knowing that this module will only be called post-process and any errors in the data structures have already been identified.
- The object passed to PrintEquationTree is the root of the equation tree

12.4.4 Access Routine Semantics

PrintIntervalList($iStruct$):

- transition: If the user interface is the command-line, write the interval $iStruct$ to cmd. If the user interface is a GUI, modify win so that the interval is displayed. In both cases, the variable name of the interval must also be displayed.

- exception: N/A

PrintEquationTree(*eStruct*):

- transition: If the user interface is the command-line, write the equation tree represented by *eStruct* to cmd. If the user interface is a GUI, modify win so that the equation tree is displayed.
- exception: N/A

13 MIS of the Interval Data Structure Module

13.1 Module

IntervalStruct

13.2 Uses

N/A

13.3 Syntax

13.3.1 Exported Access Programs

Name	In	Out	Exceptions
IntervalStruct	$String$, $\mathbb{R}^2$	$intervalStruct$	IV_ORD_VIOLATED
GetVariableName	-	$String$	-
GetMinBound	-	$\mathbb{R}$	-
GetMaxBound	-	$\mathbb{R}$	-
SetVariableName	$String$	-	-
SetMinBound	$\mathbb{R}$	-	IV_ORD_VIOLATED
SetMaxBound	$\mathbb{R}$	-	IV_ORD_VIOLATED

13.4 Semantics

13.4.1 State Variables

# For R2 using DD1

- $variableName$: $String$
- $minBound : \mathbb{R}$
- $maxBound : \mathbb{R}$

13.4.2 Access Routine Semantics

IntervalStruct($varName, minB, maxB$):

- output: $out := newInterval$
- transition: Update state variables $variableName$, $minBound$, and $maxBound$ with the provided values $varName$, $minB$, and $maxB$
- exception: $exc :=$
  $(minB > maxB \Rightarrow IV\_ORD\_VIOLATED)$

GetVariableName():

- output: $out := variableName$
- exception: N/A

GetMinBound():

- output: $out := minBound$
- exception: N/A

GetMaxBound():

- output: $out := maxBound$
- exception: N/A

SetVariableName($varName$):

- transition: Update state variable $variableName$ with the provided value $varName$
- exception: N/A

SetMinBound($minB$):

- transition: Update state variable $minBound$ with the provided value $minB$
- exception: $exc :=$
  $(minB > maxBound \Rightarrow IV\_ORD\_VIOLATED)$

SetMaxBound($maxB$):

- transition: Update state variable $maxBound$ with the provided value $maxB$
- exception: $exc :=$
  $(maxB < minBound \Rightarrow IV\_ORD\_VIOLATED)$

14 MIS of the Equation Data Structure Module

14.1 Module

EquationStruct

14.2 Uses

N/A

14.3 Syntax

14.3.1 Exported Access Programs

Name	In	Out	Exceptions
EquationStruct	$String^2$, $equationStruct^2$	$equationStruct$	-
GetOperator	-	$String$	-
GetVariableName	-	$String$	-
GetLeftOperand	-	$equationStruct$	-
GetRightOperand	-	$equationStruct$	-
SetLeftOperand	$equationStruct$	-	-
SetRightOperand	$equationStruct$	-	-

14.4 Semantics

14.4.1 State Variables

# To support R4 and R6

- $operator : String$
- $variableName : String$
- $leftOperand : equationStruct$
- $rightOperand : equationStruct$

14.4.2 Assumptions

- The decomposition of the user equation is handled by the Equation Conversion module (Section 9).
- Unsupported operators are identified and handled in the Equation Conversion module (Section 9).

- There is no setter method for the *operator* or *variableName* fields because they will not be changed after initialization.
- The values for $leftOperand$ and *rightOperand* can be set to `NULL` as required.

14.4.3 Access Routine Semantics

EquationStruct($op, vName, eStruct1, eStruct2$):

- output: $out := newEquation$
- transition: Update state variables *operator*, $variableName$, $leftOperand$, and *rightOperand* with the provided values *op*, $vName$, $eStruct1$, and $eStruct2$
- exception: N/A

GetOperator():

- output: $out := operator$
- exception: N/A

GetVariableName():

- output: $out := variableName$
- exception: N/A

GetLeftOperand():

- output: $out := leftOperand$
- exception: N/A

GetRightOperand():

- output: $out := rightOperand$
- exception: N/A

SetLeftOperand($eStruct$):

- transition: Update state variable $leftOperand$ with the provided value $eStruct$
- exception: N/A

SetRightOperand($eStruct$):

- transition: Update state variable *rightOperand* with the provided value $eStruct$
- exception: N/A

15 MIS of the Operator Data Structure Module

[Do you really want to add operators in this way? Don't you already know that your operators are addition, subtraction, multiplication and division? I like the idea of flexibility, but this might not be the right kind of flexibility. —SS]

15.1 Module

OperatorStruct

15.2 Uses

N/A

15.3 Syntax

15.3.1 Exported Access Programs

Name	In	Out	Exceptions
OperatorStruct	*String*, *Int*, *Boolean*[4]	*operatorStruct*	OP_INVALID_PRECEDENCE, OP_MISSING_OP, OP_MULTI_TYPE, OP_NO_TYPE
GetOperator	-	*String*	-
GetPrecedence	-	*Int*	-
IsUnary	-	*Boolean*	-
IsBinary	-	*Boolean*	-
IsTernary	-	*Boolean*	-
IsLeftAssociative	-	*Boolean*	-

15.4 Semantics

15.4.1 State Variables

- $operator : String$
- $precedence : Int$
- $isUnary : Boolean$
- $isBinary : Boolean$
- $isTernary : Boolean$
- $leftAssociative : Boolean$

15.4.2 Assumptions

- There are no Setter methods for this module because operator properties are fixed.
- A high integer value is associated with a high precedence operation.

15.4.3 Access Routine Semantics

OperatorStruct($op, prec, isUnary, isBinary, isTernary, isLeftAssociative$):

- output: $out := newOperator$
- transition: Update state variables $operator$, $precedence$, $isUnary$, $isBinary$, $isTernary$, and $leftAssociative$ with the provided values op, $prec$, $isUnary$, $isBinary$, $isTernary$, and $isLeftAssociative$.
- exception: $exc :=$
  $(prec < 0 \Rightarrow OP\_INVALID\_PRECEDENCE)$
  $|$
  $(op =$ “” $\Rightarrow OP\_MISSING\_OP)$
  $|$
  $((isUnary = isBinary \land isUnary = True) \lor (isUnary = isTernary \land isUnary = True) \lor (isBinary = isTernary \land isBinary = True) \Rightarrow OP\_MULTI\_TYPE)$
  $|$
  $(isUnary = isBinary = isTernary \land isUnary = False \Rightarrow OP\_NO\_TYPE)$

GetOperator():

- output: $out := operator$
- exception: N/A

GetPrecedence():

- output: $out := precedence$
- exception: N/A

IsUnary():

- output: $out := isUnary$
- exception: N/A

IsBinary():

- output: $out := isBinary$
- exception: N/A

IsTernary():

- output: $out := isTernary$
- exception: N/A

IsLeftAssociative():

- output: $out := leftAssociative$
- exception: N/A

References

Carlo Ghezzi, Mehdi Jazayeri, and Dino Mandrioli. *Fundamentals of Software Engineering*. Prentice Hall, Upper Saddle River, NJ, USA, 2nd edition, 2003.

Daniel M. Hoffman and Paul A. Strooper. *Software Design, Automated Testing, and Maintenance: A Practical Approach*. International Thomson Computer Press, New York, NY, USA, 1995. URL `http://citeseer.ist.psu.edu/428727.html`.

16 Appendix

Table 2: Possible Error Exceptions

Message ID	Error Message
EQC_NO_OPS	Error: No operators were passed to the equation conversion module.
EQC_INVALID_OP	Error: The equation conversion module cannot parse the passed operator.
EQC_UNBALANCED_TERMINATOR	Error: An unbalanced terminator was passed to the equation conversion module.
EQC_UNSUPPORTED_OP	Error: The user equation contains an unsupported operator. Supported operators include $< supportedOperators >$.
EQC_INCOMPLETE_OP	Error: An operator was found that does not have sufficient operands.
IN_BAD_FILE	Error: The file could not be read.
IN_EMPTY_FILE	Error: The file was empty.
IVC_CONV_ERR_MIN	Error: The string provided for the minimum bound cannot be converted to a real number.
IVC_CONV_ERR_MAX	Error: The string provided for the maximum bound cannot be converted to a real number.
IVC_EMPTY_VARNAME	Error: Intervals must have an associated variable name.
IVC_NO_BOUNDS	Error: No values provided for either interval bound.
OP_INVALID_PRECEDENCE	Error: Cannot assign a precedence value less than 0.
OP_MISSING_OP	Error: Cannot have an operator with no representative symbol.
OP_MULTI_TYPE	Error: An operator cannot be overloaded to be unary, binary, and ternary.
OP_NO_TYPE	Error: Operators must be assigned a number of operands type.
SOL_UNSUPPORTED_OP	Error: An unsupported operation was encountered while solving for the range of the equation.
VC_INVALID_VARNAME	Error: Encountered a variable name with reserved characters $(+, -, *, \hat{}, (,))$.
VC_MISSING_VARS	Error: A variable is referenced in the user equation that does not exist in the variable list.

VC_NO_FUNC	Error: No user equation was received.

Table 3: Possible Warning Exceptions

Message ID	**Error Message**
EQ_WRONG_OPERATOR_TYPE	Warning: The operator must have type *string*. String type conversion has been applied.
EQ_WRONG_VARNAME_TYPE	Warning: The variable name must have type *string*. String type conversion has been applied.
EQC_CONST_FUNC	Warning: The user equation is a constant value and the range will only include this value.
EQC_IMPLICIT_MULT	Warning: Encountered an implicit multiplication of a constant value and a variable. Expanding with explicit operator.
IV_ORD_VIOLATED	Warning: Value provided for intervals are not in increasing order. The values have been exchanged to maintain the interval ordering.
IVC_NO_MIN	Warning: No minimum interval bound given. Setting it to the same value as the maximum bound.
IVC_NO_MAX	Warning: No maximum interval bound given. Setting it to the same value as the minimum bound.
VC_EXTRA_VARS	Warning: There are more variables in the variable list than the user equation. Extraneous variables will be ignored.