

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

Geneva Smith

December 8, 2017

1 Revision History

Date	Version	Notes
December 2017	7, 1.1.1	Revised the operator data structure with missing “get” operator, a new exception, and seperated the <code>numOperands</code> Integer state variable into three Boolean state variables; added terminator variables to Solver module
December 2017	5, 1.1	Added a specification for an operator data structure
November 2017	27, 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	1
5	Module Decomposition	2
6	MIS of the Control Flow Module	3
6.1	Module	3
6.2	Uses	3
6.3	Syntax	3
6.3.1	Exported Access Programs	3
6.4	Semantics	3
6.4.1	State Variables	3
6.4.2	Access Routine Semantics	3
7	MIS of the User Input Module	5
7.1	Module	5
7.2	Uses	5
7.3	Syntax	5
7.3.1	Exported Access Programs	5
7.4	Semantics	5
7.4.1	Environment Variables	5
7.4.2	State Variables	5
7.4.3	Assumptions	5
7.4.4	Access Routine Semantics	6
8	MIS of the Interval Conversion Module	7
8.1	Module	7
8.2	Uses	7
8.3	Syntax	7
8.3.1	Exported Access Programs	7
8.4	Semantics	7
8.4.1	State Variables	7
8.4.2	Assumptions	7
8.4.3	Access Routine Semantics	7

9	MIS of the Equation Conversion Module	9
9.1	Module	9
9.2	Uses	9
9.3	Syntax	9
9.3.1	Exported Access Programs	9
9.4	Semantics	9
9.4.1	State Variables	9
9.4.2	Assumptions	9
9.4.3	Access Routine Semantics	9
10	MIS of the Variable Consolidation Module	11
10.1	Module	11
10.2	Uses	11
10.3	Syntax	11
10.3.1	Exported Access Programs	11
10.4	Semantics	11
10.4.1	State Variables	11
10.4.2	Assumptions	11
10.4.3	Access Routine Semantics	11
11	MIS of the Range Solver Module	13
11.1	Module	13
11.2	Uses	13
11.3	Syntax	13
11.3.1	Exported Constants	13
11.3.2	Exported Access Programs	13
11.4	Semantics	13
11.4.1	State Variables	13
11.4.2	Assumptions	13
11.4.3	Access Routine Semantics	13
12	MIS of the Output Module	15
12.1	Module	15
12.2	Uses	15
12.3	Syntax	15
12.3.1	Exported Access Programs	15
12.4	Semantics	15
12.4.1	State Variables	15
12.4.2	Environment Variables	15
12.4.3	Assumptions	15
12.4.4	Access Routine Semantics	15

13 MIS of the Interval Data Structure Module	17
13.1 Module	17
13.2 Uses	17
13.3 Syntax	17
13.3.1 Exported Access Programs	17
13.4 Semantics	17
13.4.1 State Variables	17
13.4.2 Access Routine Semantics	17
14 MIS of the Equation Data Structure Module	19
14.1 Module	19
14.2 Uses	19
14.3 Syntax	19
14.3.1 Exported Access Programs	19
14.4 Semantics	19
14.4.1 State Variables	19
14.4.2 Assumptions	19
14.4.3 Access Routine Semantics	20
15 MIS of the Operator Data Structure Module	21
15.1 Module	21
15.2 Uses	21
15.3 Syntax	21
15.3.1 Exported Access Programs	21
15.4 Semantics	21
15.4.1 State Variables	21
15.4.2 Assumptions	22
15.4.3 Access Routine Semantics	22
16 Appendix	25

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.

Complementary documents include the System Requirement Specifications and Module Guide. The full documentation and implementation can be found at <https://github.com/GenevaS/CAS741>.

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.

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 $(c_1 \Rightarrow r_1 | c_2 \Rightarrow r_2 | \dots | c_n \Rightarrow r_n)$.

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 $(-\infty, \infty)$
Real	\mathbb{R}	Any number in $(-\infty, \infty)$
String	$char^n$	A sequence of alphanumeric and special characters

The specification of Companion Cube Calculator uses some derived data types: sequences, strings, and tuples. Sequences are lists filled with elements of the same data type. Strings are sequences of characters. Tuples contain a list of values, potentially of different types. 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

6.1 Module

Main

6.2 Uses

Input (Section 7), 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
Main	-	-	-

6.4 Semantics

6.4.1 State Variables

N/A

6.4.2 Access Routine Semantics

Main():

- transition: Create data structures to contain the user inputs and modify their states for the Output module.

```
# Get User Input
userInputs := Input.GetUserInputs()
eqString := userInputs[0]
varList := userInputs[1, userInputs.Length - 1]

# Convert input into equation and interval data structures using
# the list of valid operators from Solver
operators := Solver.GetValidOperators()
Consolidate.ConvertAndCheckInputs(eqString, varList, operators)

# Get the equation and interval data structures and pass them to
# the Solver module
equationData := Consolidate.GetEquationStruct()
intervalDataList := Consolidate.GetIntervalStructList()
```

```
range := Solver.CalculateRange(equationData, intervalDataList)

# Report the results back to the user
Output.PrintInterval(range)
Output.PrintEquationTree(equationData)
for each interval in intervalDataList:
    Output.PrintInterval(interval)
```

7 MIS of the User Input Module

[Should GetUserInputs take a string as 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	-	<i>String</i> 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: $exc :=$
 $(\neg inputFile \vee \neg Read(inputFile) \Rightarrow IN_BAD_FILE)$
 $|$
 $(Read(inputFile) == \emptyset \Rightarrow IN_EMPTY_FILE)$

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	<i>String</i> ³	<i>intervalStruct</i>	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: $exc :=$
($varName = "" \Rightarrow IVC_EMPTY_VARNAME$)
|
($ToReal(min) \notin \mathbb{R} \Rightarrow IVC_CONV_ERR_MIN$)

$$\begin{aligned} &| \\ &(ToReal(max) \notin \mathbb{R} \Rightarrow IVC_CONV_ERR_MAX) \\ &| \\ &(min = max = "" \Rightarrow IVC_NO_BOUNDS) \\ &| \\ &(min = "" \wedge max \neq "" \Rightarrow IVC_NO_MIN) \\ &| \\ &(min \neq "" \wedge max = "" \Rightarrow IVC_NO_MAX) \end{aligned}$$

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 Access Programs

Name	In	Out	Exceptions
MakeEquationTree	<i>String</i> , <i>operatorStruct</i> ⁿ	<i>equationStruct</i>	EQC_CONST_FUNC, EQC_IMPLICIT_MULT, EQC_INCOMPLETE_OP, EQC_UNSUPPORTED_OP
GetVariableList	-	<i>String</i> ⁿ	-

9.4 Semantics

9.4.1 State Variables

- *variableList* : *String*ⁿ

9.4.2 Assumptions

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

9.4.3 Access Routine Semantics

MakeEquationTree(*userEquation*, *supportedOperations*):

- 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*) ∈ ℝ ⇒ EQC_CONST_FUNC)
|
(∃ *subEq* | *subEq* = {*subEq*₁, *subEq*₂} ∧ *subEq*₁ ∈ ℝ ∧ *subEq*₂ ∈ *variableList*)

$\Rightarrow EQC_IMPLICIT_MULT)$

|

$(\exists op \in userEquation | (NULL < op > userEquation) \vee (userEquation < op > NULL) \Rightarrow EQC_INCOMPLETE_OP)$

|

$(\exists op | op \in userEquation \wedge op \notin supportedOperations \Rightarrow EQC_UNSUPPORTED_OP)$

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	<i>String</i> , <i>String</i> ^{<i>n</i>} , <i>operatorStruct</i> ^{<i>n</i>}	-	VC_MISSING_VARS, VC_EXTRA_VARS, VC_NO_FUNCTION, VC_INVALID_VARNAME
GetEquationStruct	-	<i>equationStruct</i>	-
GetIntervalStructList	-	<i>intervalStruct</i> ^{<i>n</i>}	-

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: $exc :=$
 $(\exists var | var \in eqString \wedge var \notin varList \Rightarrow VC_MISSING_VARS)$
 $|$
 $(\exists var | var \notin eqString \wedge var \in varList \Rightarrow VC_EXTRA_VARS)$
 $|$
 $(eqString == "" \Rightarrow VC_NO_FUNCTION)$
 $|$
 $(\exists varName \supset \{+, -, *, ^, (,)\} \Rightarrow VC_INVALID_VARNAME)$

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* : *operatorStructⁿ*
- *supportedTerminators* : *Stringⁿ*

11.3.2 Exported Access Programs

Name	In	Out	Exceptions
GetValidOperators	-	<i>operatorStructⁿ</i>	-
GetValidTerminators	-	<i>Stringⁿ</i>	-
FindRange	<i>equationStruct</i> , <i>intervalStructⁿ</i>	<i>intervalStruct</i>	SOL_UNSUPPORTED_OP

11.4 Semantics

11.4.1 State Variables

N/A

11.4.2 Assumptions

- The type of *intervalStructⁿ* 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	<i>intervalStruct</i>	-	-
PrintEquationTree	<i>equationStruct</i>	-	-

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	$String, \mathbb{R}^2$	<i>intervalStruct</i>	IV_ORD_VIOLATED
GetVariableName	-	<i>String</i>	-
GetMinBound	-	\mathbb{R}	-
GetMaxBound	-	\mathbb{R}	-
SetVariableName	<i>String</i>	-	-
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	<i>String, Int, Boolean</i> ⁴	<i>operatorStruct</i>	OP_MISSING_OP, OP_MULTI_TYPE, OP_NO_TYPE
GetOperator	-	<i>String</i>	-
GetPrecedence	-	<i>Int</i>	-
IsUnary	-	<i>Boolean</i>	-
IsBinary	-	<i>Boolean</i>	-
IsTernary	-	<i>Boolean</i>	-
IsLeftAssociative	-	<i>Boolean</i>	-

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 low 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* :=
(*op* = "" \Rightarrow *OP_MISSING_OP*)
|
((*isUnary* = *isBinary* \wedge *isUnary* = *True*) \vee (*isUnary* = *isTernary* \wedge *isUnary* = *True*) \vee (*isBinary* = *isTernary* \wedge *isBinary* = *True*) \Rightarrow *OP_MULTI_TYPE*)
|
(*isUnary* = *isBinary* = *isTernary* \wedge *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_UNSUPPORTED_OP	Error: The user equation contains an unsupported operator. Supported operators include <i>< supportedOperators ></i> .
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_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 (+, -, *, ^, (,)).
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 <i>string</i> . String type conversion has been applied.
EQ_WRONG_VARNAME_TYPE	Warning: The variable name must have type <i>string</i> . 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.