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

Geneva Smith

November 27, 2017

1 Revision History

Date	Version	Notes
	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

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.

4 Notation

[You should describe your notation. You can use what is below as a starting point. —SS]

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
character	char	a single symbol or digit
integer	$\mathbb{Z}$	a number without a fractional component in (-∞, ∞)
natural number	$\mathbb{N}$	a number without a fractional component in [1, ∞)
real	$\mathbb{R}$	any number in (-∞, ∞)

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

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

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 $varName, minBound, maxBound$
- 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: $exc :=$
  $(\nexists inputFile \vee \neg Read(inputFile) \Rightarrow IN\_BAD\_FILE)$
  $|$
  $(Read(inputFile) == \emptyset \Rightarrow IN\_EMPTY\_FILE)$

8 MIS of the Interval Conversion Module

8.1 Module

IntervalConversion

8.2 Uses

IntervalStruct (Section 13)

8.3 Syntax

8.3.1 Exported Access Programs

Name	In	Out	Exceptions
MakeInterval	$String^2$	$intervalStruct$	IVC_CONV_ERR_MIN, IVC_CONV_ERR_MAX, 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(min, max):

- output: $out := newInterval$
- exception: $exc :=$
  $(ToReal(min) \notin \mathbb{R} \Rightarrow IVC\_CONV\_ERR\_MIN)$
  |
  $(ToReal(max) \notin \mathbb{R} \Rightarrow IVC\_CONV\_ERR\_MAX)$
  |
  $(min == \text{“”} \Rightarrow IVC\_NO\_MIN)$
  |
  $(max == \text{“”} \Rightarrow IVC\_NO\_MAX)$

9 MIS of the Equation Conversion Module

9.1 Module

EquationConversion

9.2 Uses

EquationStruct (Section 14)

9.3 Syntax

9.3.1 Exported Access Programs

Name	In	Out	Exceptions
MakeEquationTree	$String$, $String^n$	$equationStruct$	EQC_CONST_FUNC, EQC_IMPLICIT_MULT, EQC_INCOMPLETE_OP, EQC_UNSUPPORTED_OP
GetVariableList	-	$String^n$	-

9.4 Semantics

9.4.1 State Variables

- $variableList : String^n$

9.4.2 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: $out := equationTreeRoot$
- exception: $exc :=$
  $(ToReal(userEquation) \in \mathbb{R} \Rightarrow EQC\_CONST\_FUNC)$
  |
  $(\exists subEq | subEq = \{subEq_1, subEq_2\} \wedge subEq_1 \in \mathbb{R} \wedge subEq_2 \in variableList$
  $\Rightarrow EQC\_IMPLICIT\_MULT)$
  |
  $(\exists op \in userEquation | (NULL < op > userEquation) \vee (userEquation < op > NULL) \Rightarrow EQC\_INCOMPLETE\_OP)$
  |
  $(\exists op | op \in userEquation \wedge op \notin supportedOperations \Rightarrow EQC\_UNSUPPORTED\_OP)$

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	$String$, $String^n$, $String^n$	-	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.
- 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)

11.3 Syntax

11.3.1 Exported Constants

- $supportedOps : String^n$

11.3.2 Exported Access Programs

Name	In	Out	Exceptions
GetValidOperators	-	$String^n$	-
FindRange	$equationStruct$, $intervalStruct^n$	$intervalStruct$	SOL_INSUFF_PARAMS, SOL_WRONG_EQ_TYPE, SOL_WRONG_IV_TYPE, 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

FindRange($eStruct, ivStructList$):

- output: $out := intervalStruct$

- exception: $exc :=$
  $(\nexists eStruct \vee \nexists ivStructList \Rightarrow SOL\_INSUFF\_PARAMS)$
  $|$
  $(eStruct \neq equationStruct \Rightarrow SOL\_WRONG\_EQ\_TYPE)$
  $|$
  $(ivStructList \neq intervalStruct^n \vee ivStructList \neq NULL$
  $\Rightarrow SOL\_WRONG\_IV\_TYPE)$
  $|$
  $((\exists op \in eStruct \wedge op \notin supportedOps)$
  $\vee (\exists iv1, iv2 \in ivStructList \wedge \nexists op \in supportedOps | op(iv1, iv2) \vee op(iv2, iv1))$
  $\Rightarrow UNSUPPORTED\_OP)$

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	$\mathbb{R}^2$	*intervalStruct*	IV_ORD_VIOLATED
GetMinBound	-	$\mathbb{R}$	-
GetMaxBound	-	$\mathbb{R}$	-
SetMinBound	$\mathbb{R}$	-	IV_ORD_VIOLATED
SetMaxBound	$\mathbb{R}$	-	IV_ORD_VIOLATED

13.4 Semantics

13.4.1 State Variables

For R2 using DD1

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

13.4.2 Access Routine Semantics

IntervalStruct($minB, maxB$):

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

GetMinBound():

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

GetMaxBound():

- output: $out := maxBound$
- 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*	EQ_INSUFF_PARAMS, EQ_WRONG_VARNAME_TYPE, EQ_WRONG_OPERATOR_TYPE, EQ_WRONG_OPERAND_TYPE
GetOperator	-	*String*	-
GetVariableName	-	*String*	-
GetLeftOperand	-	*equationStruct*	-
GetRightOperand	-	*equationStruct*	-
SetLeftOperand	*equationStruct*	-	EQ_INSUFF_PARAMS, EQ_WRONG_OPERAND_TYPE
SetRightOperand	*equationStruct*	-	EQ_INSUFF_PARAMS, EQ_WRONG_OPERAND_TYPE

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* field because it 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: $out := equationStruct$
- transition: Update state variables $operator$, $variableName$, $leftOperand$, and $rightOperand$ with the provided values op, $vName$, $eStruct1$, and $eStruct2$
- exception: $exc :=$
  $(\nexists op \vee \nexists vName \vee \nexists eStruct1 \vee \nexists eStruct2 \Rightarrow EQ\_INSUFF\_PARAMS)$
  $|$
  $(vName \neq string \Rightarrow EQ\_WRONG\_VARNAME\_TYPE)$
  $|$
  $(op \neq string \Rightarrow EQ\_WRONG\_OPERATOR\_TYPE)$
  $|$
  $(eStruct1 \neq equationStruct \vee eStruct1 \neq NULL \vee eStruct2 \neq equationStruct \vee eStruct2 \neq NULL \Rightarrow EQ\_WRONG\_OPERAND\_TYPE)$

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: $exc :=$
  $(\nexists eStruct \Rightarrow EQ\_INSUFF\_PARAMS)$
  $|$
  $(eStruct \neq equationStruct \Rightarrow EQ\_WRONG\_OPERAND\_TYPE)$

SetRightOperand($eStruct$):

- transition: Update state variable $rightOperand$ with the provided value $eStruct$
- exception: $exc :=$
  $(\nexists eStruct \Rightarrow EQ\_INSUFF\_PARAMS)$
  $|$
  $(eStruct \neq equationStruct \Rightarrow EQ\_WRONG\_OPERAND\_TYPE)$

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.

15 Appendix

Table 2: Possible Error Exceptions

Message ID	**Error Message**
EQ_INSUFF_PARAMS	Error: Insufficient number of parameters provided to the equation data structure.
EQ_WRONG_OPERAND_TYPE	Error: Operands must have type *equationStruct*.
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.
SOL_INSUFF_PARAMS	Error: Insufficient number of parameters provided to the range solver.
SOL_WRONG_EQ_TYPE	Error: An equation must be provided to the solver that has type *equationStruct*.
SOL_WRONG_IV_TYPE	Error: An list of intervals must be provided to the range solver. If no intervals exist, the value must be identified as `NULL`.
SOL_UNSUPPORTED_OP	Error: An unsupported operation was encountered while solving for the range of the equation.
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.
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.