Working and Processing

Data structure:

To be able to deal with nested conditions in the RAPID code, a tree data structure with nodes having the following member variables has been used –

* ConditionType – indicating which type of predicate the node’s condition is of, stored as a String.
* Condition – Stores 1 condition as a String.
* nested – Stores references to nested nodes in an ‘Array List’.
* next – Stores reference to the next node when a condition block is finished.
* fal – Stores reference to the next node accessible when the condition is false.
* prev – Refence to Parent pointer.
* prevConditon-used to store all the previous conditions leading up to a condition if any.

Whenever a conditional statement is encountered, a node is created and the condition along with its type are stored in it. The program then looks for nested conditions for the given one at hand and stores them in an Array List of that node. If an else condition is encountered a new node is created with negated condition of the if node leading up to it. Once an entire block get finished the next pointer is used to point to the next set of condition which it may encounter in code.

prev

CFGNode: String prevCondition

String ConditionType | String Condition

CFGNode fal

CFGNode next

ArrayList<CFGNode> nested

. . . .

The Tree data structure contains a root node and member functions to perform necessary operations to extract and store conditions in new nodes and store them in their appropriate places in the tree.

Example:

IF x > 10:

IF x < 5:

Array List

IF | x > 10

.

fal

.

ENDIF

IF | y == 5

IF | x < 5

ELSE

IF y == 5:

ENDIF

.

.

Algorithm:

1. Read RAPID code line by line and extract conditions from them.
2. Variables are identified and declared in z3.
3. Store the condition and its type in a CFGNode.
4. If the new condition is independent in its occurrence, a new node is created with its condition and condition type filled.

If the new condition is nested inside a conditional statement, it is put in the Array List If the new condition comes in case of a false type statement , it as pointed by a fal pointer to the if node.

1. As the code is being processed, the stack is being created and the z3 code is being generated.
2. Conditions stored in the node are passed through a parser which convert them from an infix notation to prefix notation.
3. Once the notation is ready it is added with all the previous conditions related to it and appended to the z3code output file.

False edge

True edge

If not a nested condition

Extract Conditions and create node to store it.

RAPID Code

Process:

1. Takes the code line by line

2. Depending on type of conditional statement it performs a suitable action

3. For processing, it creates a node which stores the type of condition i.e. if, else, else if, while etc.

4. For nesting of conditions node uses an array list to maintain all the nested conditions

5. Node stores all the previous conditions as well leading up to that node if it is dependent on any

6. Node keeps the track of parent pointer and a false pointer for else & else if conditions

7. To keep the track of nesting a stack is used which stores the nodes in order of their insertion and deletion

To convert rapid to Z3:

1. Z3 uses a prefix notation for its syntaxes

2. A parser file is present which is responsible to take an infix code and convert it to prefix format for z3

3. Variables are declared before they can be used

4. The code is scanned for variables and then is declared in z3

5. Conditions in the rapid code are processed according to their occurrences and their priority

6. Thus nested conditions are fused together and else conditions are negated

Data Types Mapping from Rapid to Z3

Z3 and rapid have slightly different data type support

Thus a mapping is required from rapid to z3 before we can process it

**Data types rapid and their z3 mapping:**

---------------------------------------------------------------------------------------------------------

1**. num**: num is standard data type in rapid for numbers, integer, real numbers

**Z3 mapping:**

**Int**: Int is used in rapid for integer type

**Real**: Real is used for floating point type

--------------------------------------------------------------------------------------------------------

**2**. **bool**: bool is used for Boolean data types in rapid

**Z3 mapping:**

**Bool**: Bool is used for Boolean data types in z3

---------------------------------------------------------------------------------------------------------

3. **pos**: pos is a data type used to represent a 3d point in space used mainly for robotic movements

If pos variable is cor : = [10, 20, 30]

Then we use cor.x for 10, cor.y for 20, cor.z for 30

**Z3 mapping:**

For pos, we declare 3 variables to manage the 3d point of given variable

**cor\_x** for cor.x

**cor\_y** for cor.y

**cor\_z** for cor.z

Output of Z3 solver

1. Z3 solver gives values of variables which satisfy the conditions mentioned
2. It provides with a single value for a variable
3. This value is also the smallest value near the constraints to satisfy the equation or condition
4. (get-model) is used to obtain values if there exist any
5. If any value does not exist it will generate an error with message model not available
6. (check-sat) is used to see if set of condition is satisfiable
7. (check-sat) returns yes/no answer i.e. sat/un-sat
8. Using (get-model) on a ‘no’ or ‘un-sat’ generates the error

If:

Z3 solver input: X>45

Z3 result: X=46

If:

Z3 solver input: X>50 and Y>60

Z3 result: X=51, Y=61

If:

Z3 solver input: X>50 and Y<60

Z3 result: X=51, Y=59