CS402 Introduction to Logic in Computer Science
Coursework 3: z3 SMT solver & Basic Program Verification

SoC, 20155228, Nakwon Lee

June 9, 2016

1 Getting Familiar with z3 SMT solver

1.1 Files

File name *fruit_crates_n1n2.smt2* is the solution z3 input file for problem one and two. File name *fruit_crates_n3.smt2* is the solution z3 input file for problem three.

In *fruit_crates_n1n2.smt2* file, there are two (check-sat) statements. First one is checking satisfiability when the apple is taken out from the mixed labelled crate (problem 1.1). Next one is checking satisfiability of model that is excluding the satisfiable assignment of first (check-sat) statement's result (problem 1.2).

In *fruit_crates_n3.smt2* file, there is also two (check-sat) statements. First one is checking satisfiability when the apple is taken out from the orange labelled crate. Then exclude one satisfiable assignment of first (check-sat) statement from model. Second (check-sat) is checking satisfiability of model with excluding one possible assignment.

1.2 Problem Definition for Crates

Each crate can have apples, oranges, or both. I use enumeration type to mimic crates. The type name is *RealFruits*. The domain of type *RealFruits* is {apple, orange, mixed}. Each means that the crate has apple, orange, or both. By problem definition, three variables, *realApple*, *realOrange*, and *realMixed*, must be used. They are declared as variable of type *RealFruits*. By problem definition too, three variables must be distinct over domain {apple, orange, mixed} which is represented as following SMT-LIB formula:

$$(assert\ (distinct\ realApple\ realOrange\ realMixed))$$

Last condition that is defined by problem definition is that there is no correctly labelled crate which are represented as following SMT-LIB formulas:

$$(assert\ (not\ (=\ realMixed\ mixed)))$$

$$(assert\ (not\ (=\ realApple\ apple)))$$

$$(assert\ (not\ (=\ realOrange\ orange)))$$

1.3 Problem 1: An apple is picked from the mixed labelled crate

If an apple is piked from mixed labelled crate, it means that mixed labelled crate must have only apples or both apples and oranges. This characteristic can be represented as following SMT-LIB formula:

$$(assert\ (or\ (=\ realMixed\ apple)\ (=\ realMixed\ mixed)))$$

With the constraints defined from problem definition, above formula must also be satisfied by the crates model.

Consequently, if there is an assignment to satisfy all constraints over three variables, *realApple*, *realOrange*, and *realMixed*, that assignment is the possible crates result. The SMT-LIB formula (check-sat) is added to check the satisfiability of model. To find a possible assignment, add SMT-LIB formula (get-model) next to the (check-sat) formula.

As a result, the possible assignment is produced by z3. The assignment is follows:

$$(= \ realApple \ orange)$$

$$(= \ realOrange \ mixed)$$

$$(= \ realMixed \ apple)$$

1.4 Problem 2: Is the possible assignment is only one solution for problem 1?

To check whether the possible assignment of problem 1, I modify the model as excluding the found assignment. To eliminate the assignment from model, following constraints are added:

$$(assert \ (not \ (= \ realApple \ orange)))$$

$$(assert \ (not \ (= \ realOrange \ mixed)))$$

$$(assert \ (not \ (= \ realMixed \ apple)))$$

And then, to check the satisfiability and possible assignment, add (check-sat) and (get-model) formulas. With modified model, z3 returns the unsat result which means that there is no possible assignment for model. It means that the possible assignment produced by problem 1 model is the only one possible assignment for crates model.

1.5 Problem 3: An apple is picked from the orange labelled crate

If an apple is piked from orange labelled crate, it means that orange labelled crate must have only apples or both apples and oranges. This characteristic can be represented as following SMT-LIB formula:

$$(assert \ (or \ (= \ realOrange \ apple) \ (= \ realOrange \ mixed)))$$

With the constraints defined from problem definition, above formula must also be satisfied by the crates model.

Consequently, if there is an assignment to satisfy all constraints over three variables, *realApple*, *realOrange*, and *realMixed*, that assignment is the possible crates result. The SMT-LIB formula (check-sat) is added to check the satisfiability of model. To find a possible assignment, add SMT-LIB formula (get-model) next to the (check-sat) formula.

As a result, the possible assignment is produced by z3 for problem 3. The assignment is follows:

$$(= \ realApple \ orange)$$

$$(= \ realOrange \ mixed)$$

$$(= \ realMixed \ apple)$$

As the same way of problem 2, model is refined by eliminating produced assignment. Because produced assignment is same as problem 1, the additional constraint formula for eliminating assignment is also same as problem 2 as follows:

$$(assert \ (not \ (= \ realApple \ orange)))$$

$$(assert\ (not\ (=\ realOrange\ mixed)))$$

$$(assert\ (not\ (=\ realMixed\ apple)))$$

Different from result of problem 2, even in modified model, it is possible to assign values satisfying constraints. That additional assignment is follows

$$(=\ realApple\ mixed)$$

$$(=\ realOrange\ apple)$$

$$(=\ realMixed\ orange)$$

Consequently, if an apple is picked from orange labelled crate, there is more than one possible assignment.

Note that the first produced assignment is not decidable because z3 produces one of the possible assignment. Therefore, the order of produced assignment can be changed.

2 Program Verification

2.1 Execution Guide

IMPORTANT! The verifier file *verifier.py* must be in same directory with soot.sh and target java file. It is recommended to move the *verifier.py* file to soot directory.

Execution command is follows:

$$\$\ python\ verifier.py\ [Test.java]$$

2.2 Function: verify

This is the main function for my verification tool. In verify function, initiation of files and converting java to SSA are conducted. After initialization, main loop is working until reaching the end of shimple file. And then, end of main loop, the actual verification of given program is conducted by calling the external SMT-solver z3.

2.3 SSA converting

Given java program is converted to shimple program, which is the form of Single Static Assignment. Generated shimple file name is AAA.shimple when the given java file name is AAA.java. Converting to SSA form is conducted by calling the external tool soot.

2.4 smt2 file initialization

The file verifile.smt2 is the intermediate file that is used as input for z3 solver. During the verification program execution, constraints, which represent the given java file's model, are written to the verifile.smt2 file. And then the file is used as input for z3 solver.

2.5 Label condition system

I define the set of SSA instructions using label. The current label (it means that the current line is in the block of current label) is the current block, which is the set of continuous instructions. And conditions (formulas) are mapped to each label. It semantically means that the label is executed if the corresponding label condition is satisfied. The variable labelconddic, which is dictionary structure type, is the storage for label and its condition. Therefore, the condition of label is modified by the branching instructions such as if and goto instruction.

2.6 Concept of verification

The formal concept of this verification is as follows:

$$M \models P$$

M means the given java program (exactly the testMe function) and P is the assertion property. The program make the file verifile.smt2 which include the model M and assertion property P. To find the validity of above formula, we check the satisfiability of follow:

$$M \models \neg P$$

If above formula cannot satisfiable, property P is valid in model M. Therefore, the assertion property is added with negation. And, do the actual verification using z3 solver to check the satisfiability of above formula.

2.7 Main Loop

Main loop body is repeated until reaching end of shimple file. One execution of body is the processing of one line of shimple file. At the start of loop body, the type of current line is identified and processing of the line is conducted with regard to the type of line. After the processing, current line is proceed to next line.

2.7.1 Check the type of each SSA line

function whatLine is the function that takes string of one line as input and return the type of given line as integer value. There are eleven types of lines in shimple file. If the line is not classified as one of eleven types, that line is not meaningful for this verification. The eleven types are follows:

- ParenthesisNumberLine ((?)): the line start with parenthesized integer number, which is used for Phi operation.
- SemanticAssign: the line including := operator, which can be helpful to find parameter variable.
- Assertion: the line indicating the start of assertion section (assertion condition cannot be established from only one line.)
- If statement: the line indicating if statement. it also have goto statement in same line.
- Goto statement: the line including only goto statement, which is one of indicator for end of label section
- Throw statement: the line including only throw statement, which is one of indicator for end of label section
- Label statement: the line including label statement (label statement must have only label!), which indicate the start of label section
- Type statement: the line start with type indicator, which indicate the declaration of variables
- Assignment statement: the line including = operator, which means the assignment of value
- TagetMethod statement: the line, which is start of target method (testMe)
- Return statement: the line including return statement, which indicates the end of method

The order of line type checking is important because there are some ambiguity in line types. The (?) type can be with any type of line because it is not a real operation. Therefore, it must be checked before all others. the assignment and semantic assignment is also ambiguous if the assignment is check first because := also include = symbol. Assertion is actually specific type of assignment statement so if assignment statement is classified before the assertion, assertion cannot be found. Therefore, the order of checking type must be retained as describing above.

2.7.2 Loop status indicators

Boolean variables endofmethod, inbody, inassert, and labelget are indicator of loop status. The processing task for a line is different for current status of loop. First of all, the loop is running until the endofmethod variable is false. The inbody means that current processed line is in target method (testMe). The inassert means that current processed line is in assertion section, which needs different processing mechanism from general body. The labelget means that the safe label (the label including return statement, which is not a assertion violation) is found to help producing the assertion constraint.

2.7.3 Type statement

A type statement is the declaration of variables. From the problem definition, the variables allowed are only integer type. However, the SSA form use boolean type variables to represent the intermediate status of branch condition. Therefore, the verification program must support integer and boolean types for variables.

The SMT-LIB formula for variable declaration is follows:

$$(declare - const\ \ [varname]\ \ [Type])$$

The verification program parse the Type statement line to identify variable name and Type and write the above form of variable declaration to verifile.smt2 file. (function: writeFormDVars)

2.7.4 Assignment statement

A assignment statement is assigning value to the variable. It can be represented as following SMT-LIB formula:

$$(assert\ \ (=\ \ [lefthandsidevarname]\ \ [expressionforassignedvalue]))$$

Because we use complete form of SSA, the order of formulas are not meaningful. However, the branches must be considered because an instruction is not meaningful if the state of model is not included in current block of the instruction. Therefore, I refine the above form of formula as follows:

$$(assert\ \ (=>\ \ [labelcondition]\ \ (=\ \ [lefthandsidevarname]\ \ [expressionforassignedvalue])))$$

It means that if an instruction is in a label block, the instruction must be true when the label's condition is true. But, if the label's condition is false, the instruction is not cared.

The expression for assigned value can be expression of one binary operator, or just a variable or constant. (function: writeFormAssign)

2.7.5 Assertion statement

A assertion statement is one of the assignment statement. The difference is that an assertion statement includes ¨$assertDisabled¨ string. So, I find the ¨$assertDisabled¨ string as a indicator of start of assertion area. The reason why we need to maintain assertion area is identified is that in assertion area, the branching conditions are used not only updating label conditions but also the actual assertion property of given target java program.

If the assertion statement is found and it is in the target method, inassert variable is turn to be True.

2.7.6 Target method statement

The target method statement is the start of testMe function. It is identified by the inclusion of method name. I use the < symbol as a indicator because other auto generated method must have the < symbol in name. When we find the Target method statement, turn the inbody variable as True.

2.7.7 return statement

The return statement is the indicator of end of method. Therefore, when we meet the return statement during the method body, turn the endofmethod value to false to escape the main loop.

2.7.8 Label statement

There are some ways to reach to label. First one is just proceeding to next instruction. If we meet label when we already in other label, it indicates that current label is ended and new label is started. So, we do the task corresponding to end of label block and change the current label as new label. The case of current label is exist and new label is appeared, it means that new label can be reached from the current label. Therefore, the new label's condition should be updated as follows:

$$[newlabelcond] = [newlabelcond] \ \ or \ \ [currentlabelcond]$$

Use "or" is reasonable because current label condition is one of possible way of reaching new label, not a necessary condition to reach to new label.

If the current label is already ended, just assign new label to current label.

There is special cases for label statement. If the label is the label of safe return, which means the assertion is valid, the condition of that label is used as a assertion property formula (we cannot decide the assertion property until reaching to safe label!). Therefore this safe label condition is property P and is added to verifile.smt2 file with negation.

2.7.9 If statement

An if statement is the instruction which start with "if". It means that the condition of label should be changed with regard to condition of if. The processing of if condition varies from the status of loop. If we meet if statement in inbody but not inassert status, the condition of if is used to update the condition of label that is the destination of goto. For all if statement in SSA form must have directly following goto statement, which is activated when the if condition is true. Goto is also an way of reaching label. Therefore, update the condition of destination label as follows:

$$[destlabel] = [destlabel]or([currentlabel]and[ifcond])$$

Because the destination can only be reached when the if condition is true, and current label and if condition.

The current label condition should also be updated because after the if statement, the condition of that label is same as else branch. So, update the current label as follows:

$$[currlabel] = [currlabel]and(not([ifcond]))$$

The negation is added to represent the else branch.

If the if statement is met at the inassert, it can be a indicator of safe label. Actually, at the start of assertion area, the if statement is appeared first. It is pointing the safe label or unsafe label (if the label is unsafe, goto statement for indicating safe label must be appeared next). Therefore, we check the safeness of label and set the safe label.

2.7.10 Goto statement

Goto is one of ways to reach label. Therefore, the destination label's condition should be updated as follows:

$$[newlabelcond] = [newlabelcond] \ \ or \ \ [currentlabelcond]$$

Goto is also the indicator of end of label. If we meet the goto statement, the flow must be moved to other label. So, we do the task for end of label.

The special case of goto is related to finding safe label. As I mentioned in if statement section, the first step of assertion area is the indication of safe label. If prior if label cannot specify the safe label, following goto label must indicate the safe label.

2.7.11 (?) handling

(?) is used as a condition indicator of Phi function. If we meet the (?), we add following SMT-LIB formula to verifile.smt2 file.

$$(assert \ \ (= [currlabel][tempbool?]))$$

Tempbool? means that the temporary boolean variable is true if the current label condition is true. It specify the source of execution flow when we meet the Phi function.

2.7.12 Throw statement

Throw statement is activation of assertion violation. Therefore, it is also end of label indicator.

2.7.13 Semantic assignment statement

We find the parameter variable if the variable name is semantically assigned as @parameter.

2.7.14 Handling Phi function

Phi function is special type of assignment. We add additional temporary boolean variable when we meet the (?). So, the Phi function is converted to following formula of SMT-LIB:

$$\bigwedge_{i \in conditions} (assert(= [tempbooli](= [lefthandvar][expressionforconditioni])))$$

2.8 Show Result

To show the result as VAILD and assignment of parameters, the execution result is written to the file z3output. If the result string in z3output have unsat result, the output printing of my verifier is VALID. If sat result, the assignment of parameters are printed in consol.