Coursework 2 Report

Adi Yerembessov, 20160740

June 6, 2018

1 Puzzle

1.1 Solution

I created 2 data types: crate and fruit, and a function picked which maps crates to fruits. It maps apple crates to apple fruit, orange crates to orange fruit, I didn't specify any constraints on mapping of mixed crate, so it can be mapped to any. The fact that it maps all mixed crates only to one fruit doesn't matter as we pick only once. realApple, realOrange, realMixed are crates and a model specifies their values. To decide uniqueness of a solution I get 1 model and then add that at least 1 variable should be evaluated to a different value, so it is a different model. If it becomes unsatisfiable, then the model is unique, otherwise another model is printed.

1.2 How to execute

The puzzles are executed just as any other python program:
$ python puzzleX.py where X is 1 or 2 or 3 depending on which one you want to execute.

2 Program verification

2.1 Solution

Firstly the program gets rid of the redundant lines. Then it parses line by line to create another program called child.py, at the end it imports and the produced program is being run. I made a lot of reasonable assumption in order to parse, hopefully they are correct.

2.2 Key Idea

I think lines other than if, goto, phi are just matter of parsing so it's easier to understand them just by looking at the code. The further explanation might be not clear, it's probably be better to read it and look the code simultaneously. As I said the program goes through the shimple file line by line, when it does so every goto statement(either if ... goto ... or just goto ...) creates a variable leapX, where X is an integer. During creation a leap variable is being equivalent to the negation of the condition of the if, if it's created

in an if line(e.g if x == y goto label2; then if the leap variable is called leap3 then leap3 == Not(x == y)). Therefore, a leap variable being True means jump didn't occur, if it's False jump did occur. Every leap variable is bound to a labelY, where Y is an integer. Scope of a leap variable is between it's creation and the line where its bound label leads to(i.e where the program would jump, if it executed the goto statement). A leap variable is active when we're inside its scope. Every line has its active leap variables(their quantity could be 0). Every line is implied by AND of its active variables, if it has none it's not implied, but just stated. It means if at least one active variable is False, then the line was jumped over and the implication doesn't mean anything as False $\Rightarrow$ Anything is True anyway. Otherwise the AND of the active variables is True and True $\Rightarrow$ line is equivalent to line, so in case the line wasn't jumped over its constraints are included. It might be a little confusing, so I want to clarify that a line being jumped over or not is not decided during parsing it is decided when the variables of the shimple file are evaluated, so the implications just simulate the jumps in the code. When I said that every leap is equivalent to its condition I actually lied the equivalence is also implied by AND of its active variables. Also leap variables are implied by negation of AND of their active variables, so the jump could be properly jumped over. It means if at least one active variable is False then the leap variable is True as it was jumped over. By saying properly jumped over, I meant so that after we reach the line where we jumped to, we shouldn't jump further by a line we jumped over. As there still could be some active variables, that were jumped over.

One more thing I needed to consider was the Phi node. They are actually simpler than leap variables. I assumed if the first word is something like (1) then it's one of the marks where the Phi node was reached from, word means a group of chars separated from other groups by spaces. When the program goes through a line with a mark, a new variable phiX is created, where X is an integer equal to the number in the parenthesis. And it's almost the same as leap variables. It's equivalent to its active leap variables. And the Phi node itself is parsed into phiX implies the left hand side being equal to the value besides the X mark, the lines are created for all options in the Phi node.

The last thing I needed to consider was validity. When the program parses the throw line, it parses it into negation of active leap variables implying True == False, which means that if the throw line can be not jumped over then it's not valid. Also, if the constraint of not jumping over the throw line is not satisfiable, then the line is always is jumped over, therefore the assertion is valid.

2.3 Output

As it was said in the specifications if the assertion is valid, the program prints VALID, otherwise it prints the model. Model includes leap and phi variables, they can be ignored. The name of the variables in the shimple file have the same names in the model, except $ is removed and # are replaced by _.

2.4 How to execute

The file is executed just as said in the specifications:
$ python verifier.py Test.java