Formal Methods: Implementation project

CGSC5103F

December 23, 2016

Contributors

Deniz Askin

Nipun Arora

Pratik Tiwari

1. Aim

1.1. The aim for this project is to device an algorithm which computes elements for ALT set given any sentence S. While Katzir's approach for the same is intended to work on Natural Language inputs, this algorithm focus on inputs presented in Artificial language as defined below:

a) Any propositional variable is a sentence.

b) If p and q are sentences, so are:

- ¬p
- p ∧ q
- p ∨ q

2. Theoretical framework

2.1. Katzir's Rules for ALT set in Natural Language

a) Replace any node in the tree with any of its sub-constituents

b) Replace any word in the sentence with another word from the lexicon such that the result is a well-formed sentence

2.2. Rules for ALT set in above mentioned artificial language

a) Replace any sentence T contained in S by a sentence that T contains

b) Replace ∧ by ∨

c) Replace ∨ by ∧

d) Remove negation

3. Approaches

3.1. <u>The Combinatorics Approach (Not implemented)</u>

One approach we considered is to first calculate the subset of an exhaust set which consists only of the combinations of propositions calculated by changing conjunctions to disjunctions and vice versa. For any statement containing N many connectives, there will be 2^N members of this subset.

Given that the statement in which all the propositions are combined by only conjunctions is necessarily a subset of this set, we realized that if we introduced the "Simplification" rule from propositional logic which reads: $p \wedge q \rightarrow p$ and $p \wedge q \rightarrow q$, then the statement in which the propositions are in conjunction and conjunction only could be used to derive:

- all the propositions in the statement in isolation,
- as well as the different partitions of the statement in which all the propositions are connected by conjunctions.

While implementing this approach, care needs to be taken that the simplification rule is not implemented between two propositions which are within different bracket structure. This will ensure that the rule 2.2(a) is implemented properly and a condition where a node is replaced by it's parent or sibling does not occur.

3.2. Bottom up Tree Parsing (Implemented)

The approach keeps in its center the imagination of the sentence as a tree structure. It is a branch wise bottom-up approach that starts with the lowest node of the first branch it encounters as it reads the sentence from left to right. For every sentence that it encounters, it gathers the alternatives using the rules 2.2(b), 2.2(c) and 2.2(d). We have used this approach in implementing our algorithm with Python.

4. Next steps

4.1. For approach 3.2

- While the currently implemented approach can generate most of the alternatives, what our current algorithm lacks is the ability to use the alternatives generated in one of the subordinate sentences to compute alternatives of higher. This is something we will be working on to include in future.

4.2. Top-Down Approach:

We are also of an opinion that a top-down approach would make more sense than a bottom-up approach. In cases of complicated sentence, it makes sense to first generate alternatives that come from variations in larger branches rather than those in smaller ones. This would give alternatives which are more distinct. Cognitively, this makes more sense too as it immediately gives the agent alternatives which offer world possibilities that are greater in their difference rather than those that which are different only minutely so.

Given this, we would like to come up with a slightly different implementation which does not go to the bottom most node first, but crates alternatives at the top level first and then goes on to find further alternatives by varying the sub-sentences in those alternatives.

5. Links for Code

Executable: https://repl.it/EwPS/4
GitHub handle: https://github.com/rspratik/ALTImplementation

6. Results from Current implementation (Code attached)

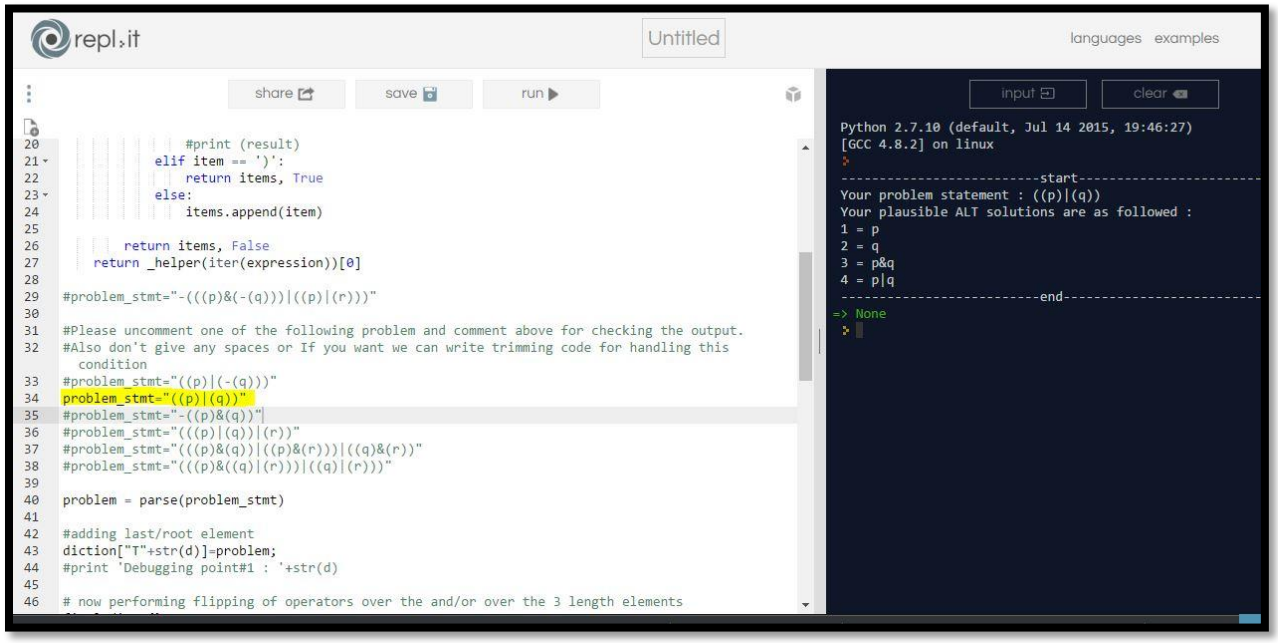

Figure 5.1. *S= ((p)|(q))*

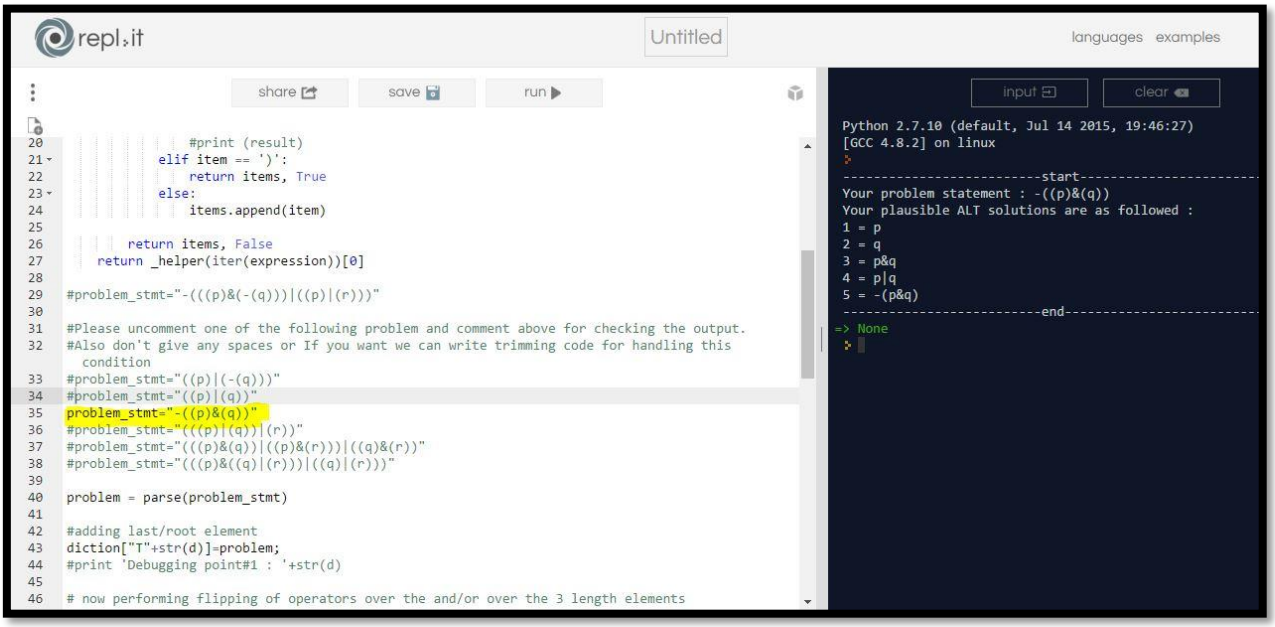

Figure 5.2. *S= ((p)&(q))*

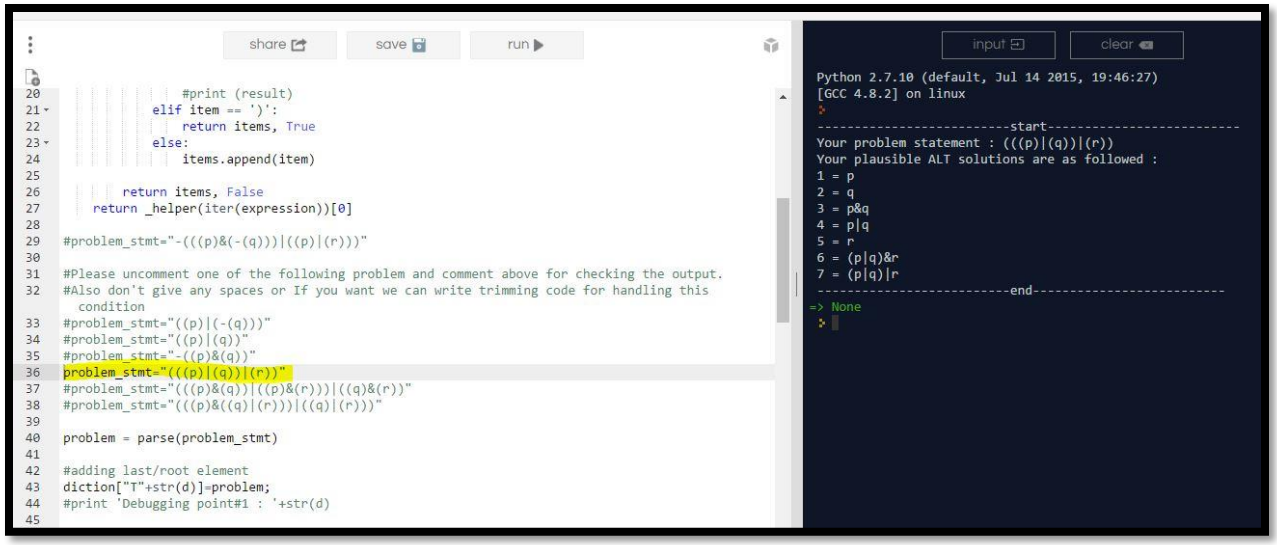

Figure 5.3 S=(((p)|(q))|(r))

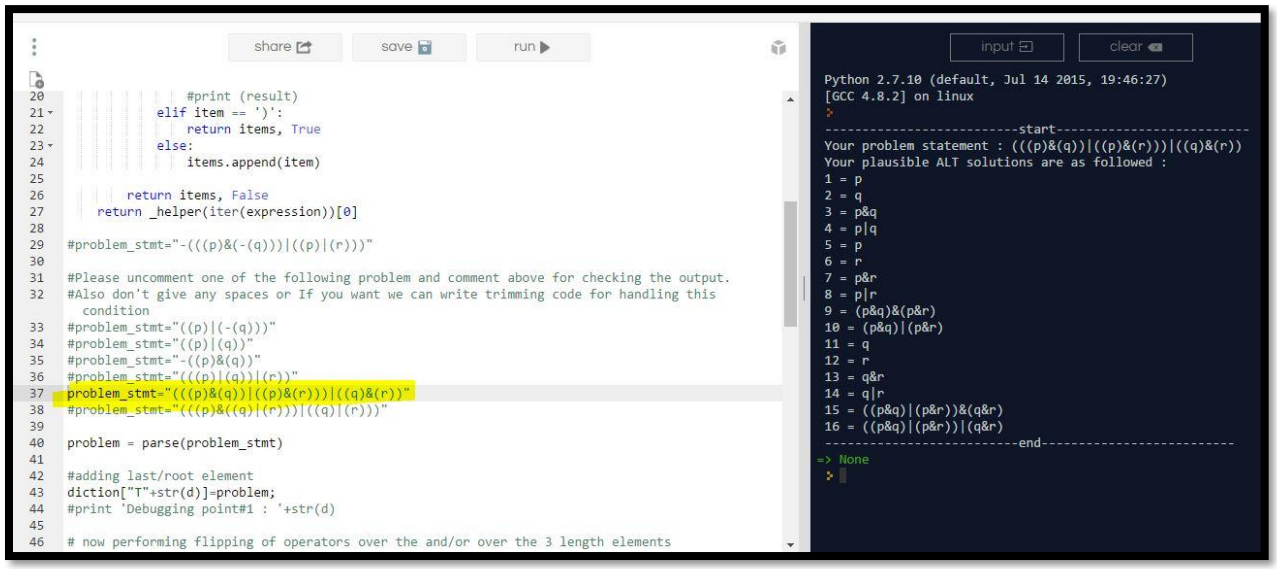

Figure 5.4 S=(((p)&(q))|((p)&(r)))|((q)&(r))

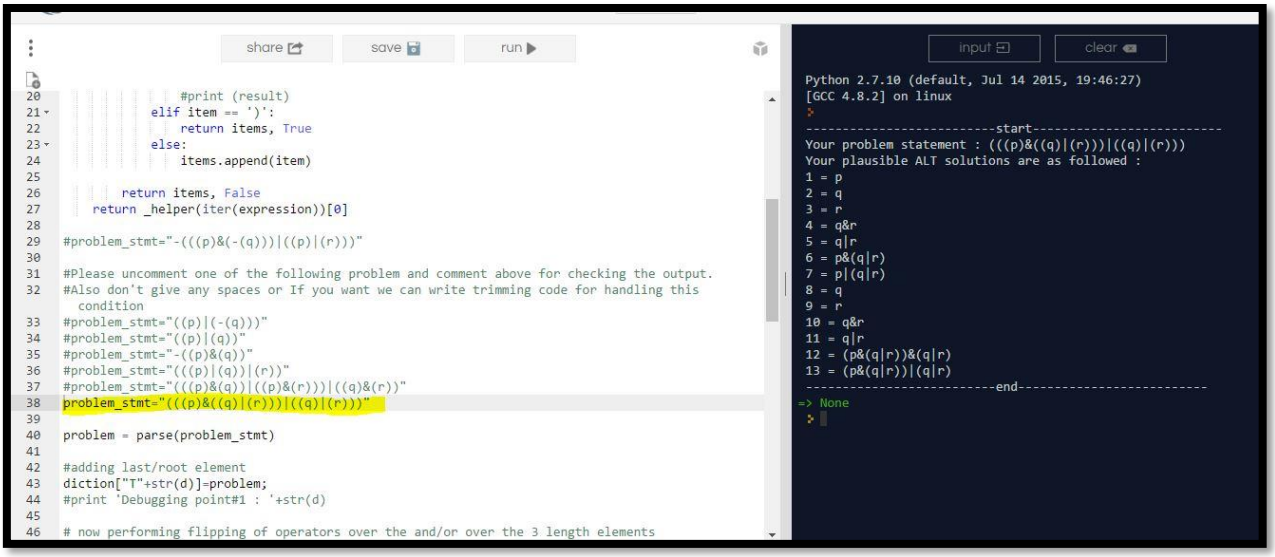

Figure 5.4 *S= (((p)&((q)|(r)))|((q)|(r)))*

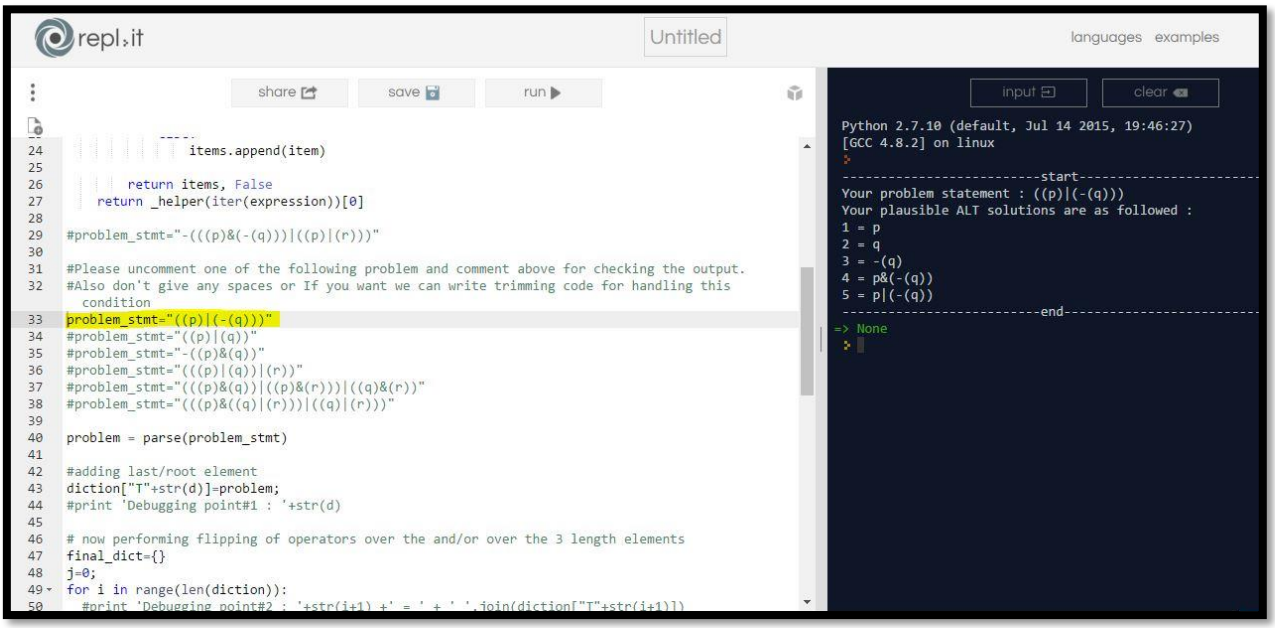

Figure 5.6 *((p)|(-(q)))*

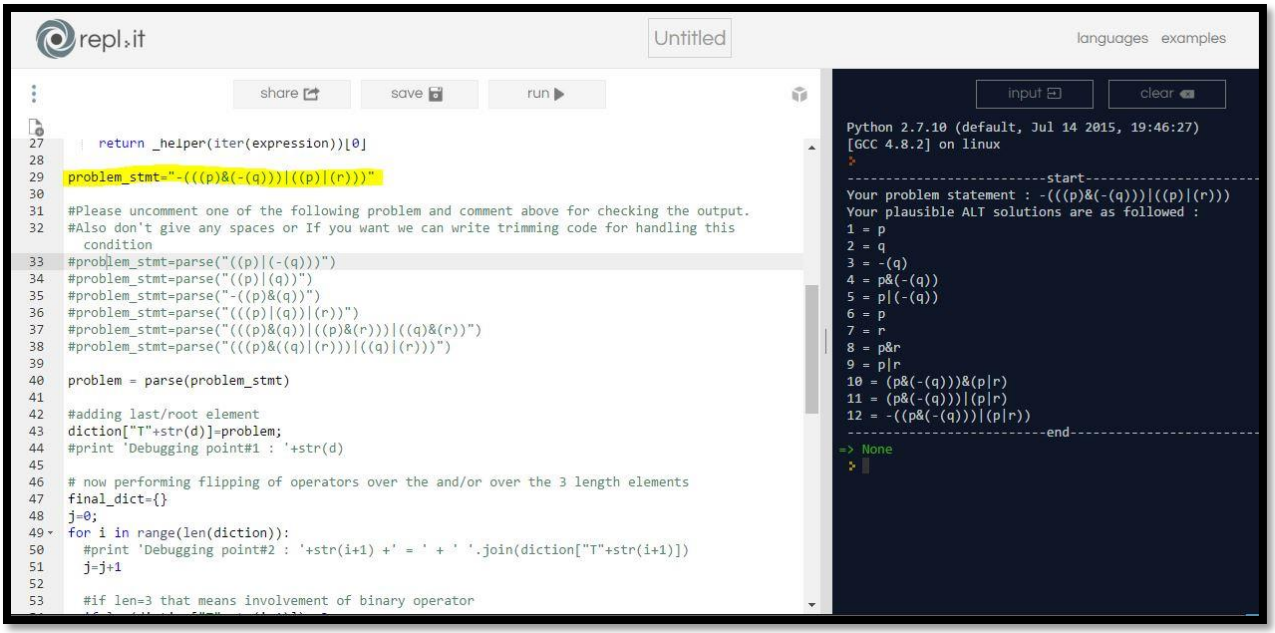

Figure 5.7 S=((p)|(q))