Pseudo Code:

```
synthesize(cfg, examples) {
    programSize=1;
    worklist = list of all terminals
    components = empty list
    while(true) {
        for (all programs in worklist) {
            if (isCorrect(program, examples)) return program
        }
        components.addAll(worklist)
        components = elimEquiv(components, examples)
        workList.clear()
        for (all operators in cfg) {
            worklist.addAll(grow(components, operator, programSize)
            worklist = elimEquiv(worklist, examples)
        }
        programSize++
    }
}
```

- **elimEquiv**: Removes observationally equivalent programs from the provided list of AST's/programs and returns the filtered list

- **grow**: Returns the enumeration of programs constructed with the given operator and combinations of its arguments from the list of components/sub programs

Features:

- Uses bottom up search
- Since the search is bottom up, the smallest program is found first
- Prunes observationally equivalent programs
- Sized based search instead of height based

Design choices:

- I switched to a bottom up approach to take advantage of the ability to eliminate observationally equivalent programs (programs that produce the same output on the given specification example inputs) as well as the reusing of subprograms

- I first attempted to implement a STUN-like procedure that would find solutions to subsets of the examples and then unify them with ‘Ite’, however I unfortunately struggled (and ran out of time) to figure out the unification process, specifically how to generate the conditions of these ‘Ite’ expressions that describe the subsets of inputs

- In an attempt to improve the search I found out that doing a size (number of AST nodes) based search rather than the traditional height based search would notably improve the scalability/efficiency as the number of available components would grow slower
  - For example when constructing programs of size 4 of the form Add(E1, E2), I only replaced pair (E1,E2) with programs such that size(E1) + size(E2) + 1 = 4, rather than simply using all possible pairs

Issues:

- **Size of program is limited:** The search usually runs out of memory or gets stuck (usually at the point where cartesian product is called to get combination of arguments during enumeration) once it reaches programs of size 9+ (at this point the list of subprograms/components is quite large)

- **Not really scalable:** Unfortunately due to the nature of bottom up synthesis, even after certain optimizations and eliminating equivalent programs, the search space still grows very quickly and wouldn’t be scalable especially for larger and more complex grammars.
  - If I succeeded with the STUN-like synthesis procedure I was trying to implement, this would have significantly improved the scalability as this technique finds solutions to smaller sub programs that work on subsets of the examples and then unifies them

Instructions to Run (Windows):

1. mvn package
2. java -cp lib/*;target/synth-1.0.jar synth.Main examples.txt -Xmx8g
   - I use the cartesian product function from the guava library, and for some reason the command with “lib;” doesn’t work, but it works with “lib/*;”

Evaluation Results:

				Phase 1	Phase 2 w/o elimEquiv	Phase 2 w/o size based search	Full Phase 2
Examples	Program	Program size (# of AST nodes)	Program height	Run time (s)	Run time (s)	Run time (s)	Run time (s)
x=2, y=2, z=3 -> 4 x=4, y=3, z=7 -> 7 x=6, y=5, z=5 -> 11 x=8, y=1, z=5 -> 9	Add(x,y)	3	1	0.005	0.016	0.017	0.021
x=2, y=2, z=3 -> 4 x=4, y=3, z=7 -> 12 x=6, y=5, z=5 -> 30 x=8, y=1, z=5 -> 8	Mult(x,y)	3	1	0.005	0.015	0.015	0.021
x=2, y=2, z=3 -> 20 x=4, y=4, z=7 -> 88 x=6, y=6, z=5 -> 132 x=8, y=1, z=5 -> 54	Multiply(Add(z, y), Add(y, x))	7	2	3.385	N/A (Error: cartesian product too large)	0.285	0.706
x=2, y=2, z=3 -> 18 x=3, y=4, z=5 -> 80 x=6, y=5, z=7 -> 245 x=3, y=6, z=5 -> 120	Multiply(z, Add(y, Multiply(y, x)))	7	3	3.268	N/A (Error: cartesian product too large)	0.185	0.629
x=2, y=2, z=3 -> 4 x=3, y=3, z=5 -> 6 x=6, y=5, z=7 -> 7 x=3, y=6, z=5 -> 5	Ite(Lt(3, y), z, Add(x, x))	8	2	N/A (ran out of memory)	N/A (Error: cartesian product too large)	0.314	12.78
x=1, y=2, z=3 -> 9 x=0, y=3, z=5 -> 15 x=2, y=4, z=7 -> 12 x=3, y=6, z=5 -> 18	Multiply(3, Ite(Lt(x, 2), z, y))	8	3	N/A (ran out of memory)	N/A (Error: cartesian product too large)	0.235	12.059
x=3, y=2, z=5 -> 6 x=5, y=3, z=5 -> 15 x=2, y=4, z=7 -> 4 x=2, y=6, z=5 -> 6	Ite(Not(Eq(x, 2)), Multiply(y, x), y)	9	3	N/A (ran out of memory)	N/A (Error: cartesian product too large)	N/A (ran out of memory)	40.972
x=3, y=2, z=5 -> 36 x=5, y=3, z=5 -> 50 x=2, y=4, z=7 -> 74 x=4, y=6, z=5 -> 78	Multiply(2, Add(z, Add(Multiply(z, y), x)))	9	4	N/A (ran out of memory)	N/A (Error: cartesian product too large)	N/A (ran out of memory)	81.965
x=3, y=2, z=5 -> 87 x=5, y=3, z=5 -> 93 x=2, y=4, z=7 -> 156 x=4, y=6, z=3 -> 42	Add(Multiply(3, Add(Multiply(z, z), x)), 3)	9	4	N/A (ran out of memory)	N/A (Error: cartesian product too large)	N/A (ran out of memory)	79.978

Evaluation Observations:

- Overall, the Phase 2 algorithm was able to synthesize faster and produce larger programs compared to phase 1
- Clearly, without the ability to eliminate observationally equivalent programs the bottom up approach is not scalable at all
- Although adding the sized based search slowed down the synthesis time quite significantly (less programs are explored each iteration of the main loop), it allowed the Phase 2 algorithm to explore further and at least reach those larger programs at size 9+

Conclusions and Additional Comments:

- Overall this project has given me a new appreciation for people who are able to use bottom up search effectively, seeing how quickly the search space grows and how difficult it is to make it scalable even for a simple grammar like the one in this project
- Although I wasn't able to implement my initial idea of using the STUN technique and my phase 2 algorithm is probably not very practical, I still learned a lot in the process and I am proud that I managed to implement an algorithm that was overall more efficient and useful than my phase 1 algorithm.
- When running the program, please make sure to use the following command (changed slightly from the original command):
  - java -cp lib/*;target/synth-1.0.jar synth.Main examples.txt -Xmx8g