CMPT489 PROJECT PHASE 2 REPORT

JOSEPH LUNDERVILLE (200014381)

FALL 2023 – YUEPENG WANG

PROBLEM AND DESIGN

Externally, the basic design of the Phase 2 project remains the same, solving the same basic problem as Phase 1, to synthesize a program specified in terms of input-output examples, having the following grammar (with E as the starting production):

$$E ::= Ite(B,E,E) \mid Add(E,E) \mid Multiply(E,E) \mid x \mid y \mid z \mid 1 \mid 2 \mid 3$$
$$B ::= Lt(E,E) \mid Eq(E,E) \mid And(B,B) \mid Or(B,B) \mid Not(B)$$
$$x, y, z \in \textbf{\textit{Variables}} \quad 1,2,3 \in \textbf{\textit{Constants}}$$

E-productions denote expressions evaluating to a Java `int`, and B-productions denote expressions evaluating to a Java `boolean`, so the input-output examples are always in the form of an integer substitution for x, y, z, with an expected integer output. The semantic interpretation follows the most natural C-family semantics for the named operations.

INTRACTABILITY OF DEPTH

Simple enumeration performs quite well up to tree height 2, after which it runs aground catastrophically: for every increase in height, the number of possible trees us multiplied by some factor of the square of the trees of previous height.

THEORETICAL APPROACH

LINEAR SUBPROBLEM

There is a much more tractable subproblem within the overall synthesis problem consisting of programs without `Ite`. This implicitly excludes boolean condition productions, and all that remains are combinations of addition and multiplication of the input variables and positive integers.

Any program of this type is implicitly a polynomial and can be flattened into an equivalent[1] expanded sum of terms consisting of powers of x, y, and z and positive integral constants:

$$a_{0,i}+a_{1,i}\,x_i+a_{2,i}\,y_i+a_{3,i}\,z_i+a_{4,i}\,x_i^2+a_{5,i}\,x_i\,y_i+a_{6,i}\,x_i\,z_i+a_{7,i}\,y_i^2\ldots a_{n,i}\,x_i^p\,y_i^q\,z_i^r=o_i$$

If the program we seek does not contain `Ite`, then, it should be representable in this form. Since each input-output example gives us values for x_i, y_i, z_i, and a corresponding expected output o_i, we can substitute these values in to develop a system of equations, and solve for the constants using integer linear programming or other constraint programming techniques like CP-SAT.

Since `Ite` can only select between different programs and does not otherwise contribute to the computation, we can partition the examples into subsets where the branch selections of the `Ite` nodes are the same, and use this analysis method on each subset independently. For this process to be practical, we need a way to efficiently discover this partitioning.

From informal exploration, it appears that if the problem is sufficiently constrained this method does seem to be effective. However, since CP-SAT and other such methods are not guaranteed to succeed in reasonable time for larger problems, and higher-order polynomials imply a potentially large number of terms, this is not a panacea.

One reason for this intractability is that absent a limit on the size of the target program, the polynomial expansion is indefinite. If we assume an arbitrary limit of height h, we can make some assumptions; there will be strictly less than 3^h leaf nodes, for example, and we can therefore be sure no term will be higher degree than 3^h – but even a precise worst case is not likely to be constraining enough to be of use. In practice, the fastest solver tested struggled with an expansion of terms beyond degree 3^1.

IMPLEMENTATION

MCMC OPTIMIZATION AND ENSEMBLE[2] DESIGN

Inspired by the paper “Stochastic Superoptimization” (Schkufza, Sharma, Aiken 2013), several applications of stochastic optimization were attempted to address the less tractable parts of this problem.

After initial success generating surprisingly good results directly generating programs exclusively using MCMC, a plan was made to structure the synthesis overall process in this way:

1. Transform the input-output examples into a set of linear algebraic equations

1 Notwithstanding integer overflow; the DSL semantics as originally implemented allow sufficiently large intermediate values to overflow, wrapping from large positive values to negative values. This occurrence breaks the monotonicity of arithmetic operations and generally makes analysis much more complex. Since this didn’t seem like an intended effect given how many symbols are required to reach that overflow condition artificially, I will treat this possibility as a deficiency in the runtime and exclude it from the analysis.

2 The ensemble synthesizer is named VoltronSynthesizer in the source.

2. Partition the examples by finding the largest possible subsets of examples whose corresponding equations can be solved together
3. Use MCMC optimization to synthesize condition subprograms to discriminate between partitions with distinct solutions
4. Unify the partial solutions using `Ite` nodes
5. Postprocess the completed solution to minimize the generated parse tree

Although results with MCMC were very promising, designing effective cost and mutation functions is a nontrivial job, even though it is relatively straightforward. Z3 proved to be sufficiently temperamental that time that was required for the full integration of methods had to be put to ensuring the reliability of the core solution method.

SOLVERS

As a last-minute attempt to work around Z3's difficulty, two separate integrations of Google OR-Tools solvers were done. The rationale for switching to OR-Tools was that the solvers included are less general and considerably less powerful, and as a practical matter, it's often the case that a less general tools is more predictable in its behaviour and more performant, *provided what you want is the specific thing it does.*

OR-Tools offers two solvers that are well suited to representing this problem: an MIP solver for mixed integer optimization problems, and a CP-SAT solver which is particularly limited, but from prior experience is good at handling large numbers of variables.

The MIP has a very straightforward interface for setting up equations as a series of linear terms summed together in an equality constraint. This solver performed better than Z3, but ultimately not enough better to make for reliable synthesis.

The CP-SAT solver is the most limited option, and unlike the other options it is *only* able to deal with integer variables. Most of its features are geared towards representing logical structures, but it is able to represent an integer weightedSum expression and equality constraint, which is the minimum needed to represent our problem.

To be clear, there was not a rigorous analysis of the performance of the different options, but anecdotally the performance and reliability improvement with OR-Tools CP-SAT was quite striking.

ENUMERATION

A number of basic measures were undertaken to improve the performance of the enumeration, such as batching synthesis jobs, reducing use of string comparison by introducing an enum type for `Symbol`s, and cacheing (and reuse) of synthesized trees during subsequent synthesis tasks.

Enumeration remains the most robust and fastest way to synthesize low complexity programs. These improvements make enumeration fast enough to be worth

attempting up to the practical limit, i.e. height 2, to rule out the simplest cases before beginning to engage more sophisticated algorithms which are likely to have high startup costs and generally are not proven to perform optimal synthesis.

FUTURE WORK AND LIMITATIONS

IF-THEN-ELSE

Currently, the synthesizer is practically limited to synthesizing programs with `Ite` nodes only for programs of height greater than 2. I.e., it can synthesize a program equivalent to

```
Ite(Lt(x, 1), Add(1, y), Multiply(y, z))
```

and

```
Add(1, Multiply(y, y)), Multiply(y, z)
```

but not

```
Ite(Lt(x, 1), Add(1, Multiply(y, y)), Multiply(y, z))
```

The systems for partitioning and synthesizing simple conditions is likely quite close to being functional, but it’s not complete at time of writing.

MINIMALITY

No attempt is made by the synthesizer to simplify or optimize its output, and it can certainly be very verbose. For example, its solution for problem `test-data/phase2-noite/0500.txt`, which is a set of examples generated from:

```
Add(Multiply(Multiply(Multiply(Add(3, y), 2), Add(y, 3)), Multiply(z, Add(y, z))), Add(Multiply(y, Add(2, 3)), x))
```

is a significant expansion:

```
Add(Add(Add(Multiply(Multiply(Multiply(Const3, Add(Const3, Const3)), VarZ), VarZ), Multiply(Multiply(Multiply(Const2, VarY), VarY), Multiply(VarZ, VarZ))), Add(VarX, Multiply(Add(Const2, Const3), VarY))), Add(Add(Multiply(Multiply(Multiply(Const3, Add(Const2, Const2)), VarY), Multiply(VarZ, VarZ)), Multiply(Multiply(Multiply(Const3, Add(Const2, Const2)), VarY), Multiply(VarY, VarZ))), Add(Multiply(Multiply(Multiply(Const3, Add(Const3, Const3)), VarY), VarZ), Multiply(Multiply(Multiply(Const2, VarY), VarY), Multiply(VarY, VarZ)))))
```

EQUIVALENCE CLASSES

There are many, many equivalent programs, and a first approach considered was to use strict equivalence to minimize enumeration time. However, there is still a deep problem of scale: an exhaustive exploration using SymPy to prove symbolic equivalence between different parse trees found approximately 36,000 equivalence

classes over 450,000 parse trees[3]. This an order of magnitude reduction, but even a squared order of magnitude is not enough to make the next generation enumerable, practically speaking.

It would have been practical to store a table of pre-generated equivalence class representatives as a substitute for computing them, and this would have definitely substantially improved performance for that step, but enumeration is only a negligible part of synthesis for more interesting problems.

TESTS

The program is tested against a suite automatically generated by a support program[4] which generates random programs in the supplied DSL, and evaluates them with arbitrarily-chosen inputs to create a test case. These programs are generated with different maximum AST depth to produce test cases of varying challenge, and with more than sufficient input-output examples (1000 distinct examples are available per test case in the *phase2-noite* suite).

To avoid the question of semantics in the face of integer overflow, the generator rejects input-output examples which produce large outputs. For test vectors representing larger programs (which tend to naturally produce larger outputs by the accumulation of addition and multiplication by positive numbers), this can constrain the input-output examples significantly.

Additionally, the test suite whose output is reproduced here is specifically testing the subset of the DSL without `Ite`, `Lt`, `Eq`, `And`, `Or`, or `Not`. The synthesizer attempts synthesis up to height 2 with a BFS enumerative synthesizer, so it is capable of solving some problems of this type, but it was deemed not important to demonstrate that capability.

Finally, only one test case is included at height 5; this is because the test run was aborted after a 2 minute timeout.

Within these strict limitations, the test suite shows 100% success for equivalent programs height 4 or less, which is a significant improvement over the phase 1 enumeration.

CONCLUSION

The phase 2 synthesizer is reliably able to synthesize a branchless subset of programs when the equivalent target program is height 4 or less. The algebraic transformation presented here reduces the complexity of this subproblem, and brings it into a form which allows easy application of standard solvers.

Despite this convenience, however, the complexity of the equivalent CP-SAT problem still appears to have issues with rapid growth in complexity beyond target programs with height 4.

3 This experiment was conducted in a Jupyter notebook which can be found in `Synth/support/EqClassGen/eqclassgen.ipynb`

4 Found in `Synth/support/ProblemGen/src/Main.hs`

There is a clear direction to incorporate synthesis of Ite branches, improving the generality of the solver, but it is not clear that such techniques will improve the handling of complexity which derives from deeper target program trees.

APPENDIX: TEST OUTPUT

```
# test-data/phase2-noite/0100.txt
[2023-12-06 07:49:02.369] INFO: Attempting solution with BFSEnum2Synthesizer
Add(Const3, VarY)
# test-data/phase2-noite/0101.txt
[2023-12-06 07:49:02.400] INFO: Attempting solution with BFSEnum2Synthesizer
Add(VarX, VarX)
# test-data/phase2-noite/0102.txt
[2023-12-06 07:49:02.410] INFO: Attempting solution with BFSEnum2Synthesizer
Add(VarX, VarX)
# test-data/phase2-noite/0103.txt
[2023-12-06 07:49:02.416] INFO: Attempting solution with BFSEnum2Synthesizer
Add(Const1, VarZ)
# test-data/phase2-noite/0104.txt
[2023-12-06 07:49:02.422] INFO: Attempting solution with BFSEnum2Synthesizer
Add(Const2, VarZ)
# test-data/phase2-noite/0105.txt
[2023-12-06 07:49:02.427] INFO: Attempting solution with BFSEnum2Synthesizer
Add(Const3, Const3)
# test-data/phase2-noite/0106.txt
[2023-12-06 07:49:02.439] INFO: Attempting solution with BFSEnum2Synthesizer
Add(VarX, VarZ)
# test-data/phase2-noite/0107.txt
[2023-12-06 07:49:02.443] INFO: Attempting solution with BFSEnum2Synthesizer
Add(Const2, Const3)
# test-data/phase2-noite/0108.txt
[2023-12-06 07:49:02.446] INFO: Attempting solution with BFSEnum2Synthesizer
Add(Const3, Const3)
# test-data/phase2-noite/0109.txt
[2023-12-06 07:49:02.451] INFO: Attempting solution with BFSEnum2Synthesizer
Add(Const1, VarX)
# test-data/phase2-noite/0200.txt
[2023-12-06 07:49:02.455] INFO: Attempting solution with BFSEnum2Synthesizer
Multiply(Add(Const3, VarY), Add(VarX, VarY))
# test-data/phase2-noite/0201.txt
[2023-12-06 07:49:04.050] INFO: Attempting solution with BFSEnum2Synthesizer
Add(Add(VarX, VarX), Add(VarY, VarY))
# test-data/phase2-noite/0202.txt
[2023-12-06 07:49:04.518] INFO: Attempting solution with BFSEnum2Synthesizer
Multiply(Add(Const2, VarY), Add(Const3, VarY))
# test-data/phase2-noite/0203.txt
[2023-12-06 07:49:05.077] INFO: Attempting solution with BFSEnum2Synthesizer
Add(Add(Const2, Const3), Multiply(Const3, Const3))
# test-data/phase2-noite/0204.txt
[2023-12-06 07:49:05.598] INFO: Attempting solution with BFSEnum2Synthesizer
Add(Add(Const1, Const1), Add(Const2, VarY))
# test-data/phase2-noite/0205.txt
[2023-12-06 07:49:05.997] INFO: Attempting solution with BFSEnum2Synthesizer
Add(Add(VarX, VarZ), VarZ)
# test-data/phase2-noite/0206.txt
[2023-12-06 07:49:06.363] INFO: Attempting solution with BFSEnum2Synthesizer
Add(Add(Const1, Const2), Add(Const3, VarZ))
# test-data/phase2-noite/0207.txt
[2023-12-06 07:49:06.712] INFO: Attempting solution with BFSEnum2Synthesizer
Add(Add(VarY, VarY), Add(VarY, VarZ))
# test-data/phase2-noite/0208.txt
[2023-12-06 07:49:07.135] INFO: Attempting solution with BFSEnum2Synthesizer
Multiply(Add(Const2, Const3), Add(VarX, VarZ))
# test-data/phase2-noite/0209.txt
[2023-12-06 07:49:07.488] INFO: Attempting solution with BFSEnum2Synthesizer
Add(Multiply(Const3, VarZ), Multiply(Const3, VarZ))
# test-data/phase2-noite/0300.txt
[2023-12-06 07:49:07.873] INFO: Attempting solution with BFSEnum2Synthesizer
Add(Add(Const3, VarY), Multiply(Const3, VarX))
# test-data/phase2-noite/0301.txt
[2023-12-06 07:49:08.234] INFO: Attempting solution with BFSEnum2Synthesizer
[2023-12-06 07:49:08.588] INFO: Attempting solution with VoltronSynthesizer
Add(Add(Const2, VarX), Add(VarY, Multiply(Const3, VarZ)))
# test-data/phase2-noite/0302.txt
[2023-12-06 07:49:08.694] INFO: Attempting solution with BFSEnum2Synthesizer
[2023-12-06 07:49:09.059] INFO: Attempting solution with VoltronSynthesizer
Add(Add(Add(Add(Const3, Const3), Multiply(Const2, VarX)), Multiply(Const3, VarY)),
Add(Multiply(VarX, VarY), Multiply(Multiply(VarX, VarY), VarZ)))
# test-data/phase2-noite/0303.txt
[2023-12-06 07:49:09.067] INFO: Attempting solution with BFSEnum2Synthesizer
[2023-12-06 07:49:09.460] INFO: Attempting solution with VoltronSynthesizer
```

```
Add(Add(Multiply(Const3, Const3), Multiply(Add(Const3, Const3), VarX)), Add(Multiply(Add(Const3,
Const3), VarY), Multiply(Multiply(Add(Const2, Const2), VarX), VarY)))
# test-data/phase2-noite/0304.txt
[2023-12-06 07:49:09.467] INFO: Attempting solution with BFSEnum2Synthesizer
[2023-12-06 07:49:09.820] INFO: Attempting solution with VoltronSynthesizer
Add(Add(Const1, Multiply(Const3, VarY)), Multiply(Const3, VarZ))
# test-data/phase2-noite/0305.txt
[2023-12-06 07:49:09.826] INFO: Attempting solution with BFSEnum2Synthesizer
[2023-12-06 07:49:10.188] INFO: Attempting solution with VoltronSynthesizer
Add(Add(Multiply(VarZ, VarZ), Multiply(Multiply(VarY, VarZ), VarZ)), Add(Multiply(Const2, VarZ),
Multiply(Multiply(Const2, VarY), VarZ)))
# test-data/phase2-noite/0306.txt
[2023-12-06 07:49:10.196] INFO: Attempting solution with BFSEnum2Synthesizer
[2023-12-06 07:49:10.560] INFO: Attempting solution with VoltronSynthesizer
Add(Add(Multiply(Multiply(Const2, VarZ), VarZ), Multiply(Const2, VarY)), Multiply(Const2, VarZ))
# test-data/phase2-noite/0307.txt
[2023-12-06 07:49:10.566] INFO: Attempting solution with BFSEnum2Synthesizer
[2023-12-06 07:49:10.921] INFO: Attempting solution with VoltronSynthesizer
Add(Add(Add(Multiply(VarY, VarY), Multiply(Multiply(Const2, VarX), VarY)), Multiply(Multiply(VarY,
VarY), VarZ)), Add(Multiply(Multiply(VarX, VarY), VarZ), Multiply(VarX, VarX)))
# test-data/phase2-noite/0308.txt
[2023-12-06 07:49:10.927] INFO: Attempting solution with BFSEnum2Synthesizer
[2023-12-06 07:49:11.292] INFO: Attempting solution with VoltronSynthesizer
[2023-12-06 07:49:11.298] WARNING: No satisfying solution found! Solver returned INFEASIBLE
Add(Add(Multiply(Const3, Add(Const3, Const3)), Multiply(Multiply(VarX, VarZ), VarZ)),
Add(Multiply(Const3, VarZ), Multiply(Multiply(Add(Const3, Const3), VarX), VarZ)))
# test-data/phase2-noite/0309.txt
[2023-12-06 07:49:11.310] INFO: Attempting solution with BFSEnum2Synthesizer
[2023-12-06 07:49:11.672] INFO: Attempting solution with VoltronSynthesizer
Add(Add(Add(Const3, Const3), Multiply(VarZ, VarZ)), Add(VarX, VarY))
# test-data/phase2-noite/0400.txt
[2023-12-06 07:49:11.677] INFO: Attempting solution with BFSEnum2Synthesizer
[2023-12-06 07:49:12.042] INFO: Attempting solution with VoltronSynthesizer
[2023-12-06 07:49:12.044] WARNING: No satisfying solution found! Solver returned INFEASIBLE
Add(Multiply(Multiply(Multiply(Multiply(Const3, Const3), VarX), VarY), Multiply(Multiply(VarY,
VarZ), VarZ)), Multiply(Multiply(Multiply(Multiply(Const3, Multiply(Const3, Const3)), VarX), VarY),
Multiply(VarZ, VarZ)))
# test-data/phase2-noite/0401.txt
[2023-12-06 07:49:12.054] INFO: Attempting solution with BFSEnum2Synthesizer
[2023-12-06 07:49:12.416] INFO: Attempting solution with VoltronSynthesizer
[2023-12-06 07:49:12.420] WARNING: No satisfying solution found! Solver returned INFEASIBLE
Add(Add(Add(Add(Multiply(Multiply(Add(Const3, Const3), VarY), VarY), Multiply(VarZ, VarZ)),
Multiply(Multiply(VarX, VarZ), VarZ)), Add(Multiply(Multiply(Const3, VarY), Multiply(VarZ, VarZ)),
Multiply(Multiply(Const3, VarX), VarY))), Add(Add(Add(Multiply(Multiply(Const2, VarY),
Multiply(VarY, VarZ)), Multiply(Multiply(VarZ, VarZ), VarZ)), Multiply(VarX, VarZ)),
Add(Multiply(Multiply(Add(Const2, Const3), VarY), VarZ), Multiply(Multiply(VarX, VarY), VarZ))))
# test-data/phase2-noite/0402.txt
[2023-12-06 07:49:12.438] INFO: Attempting solution with BFSEnum2Synthesizer
[2023-12-06 07:49:12.801] INFO: Attempting solution with VoltronSynthesizer
[2023-12-06 07:49:14.079] WARNING: No satisfying solution found! Solver returned INFEASIBLE
[2023-12-06 07:49:19.081] WARNING: No satisfying solution found! Solver returned UNKNOWN
[2023-12-06 07:49:24.084] WARNING: No satisfying solution found! Solver returned UNKNOWN
[2023-12-06 07:49:29.085] WARNING: No satisfying solution found! Solver returned UNKNOWN
[2023-12-06 07:49:30.190] WARNING: No satisfying solution found! Solver returned INFEASIBLE
[2023-12-06 07:49:35.192] WARNING: No satisfying solution found! Solver returned UNKNOWN
[2023-12-06 07:49:40.194] WARNING: No satisfying solution found! Solver returned UNKNOWN
[2023-12-06 07:49:41.158] WARNING: No satisfying solution found! Solver returned INFEASIBLE
Add(Add(Add(Multiply(Multiply(Multiply(Const3, Const3), VarY), VarY), Multiply(Multiply(Const3,
Add(Const3, Const3)), VarX)), Add(Multiply(Multiply(Const3, Add(Const3, Const3)), VarY),
Multiply(Multiply(Multiply(Const3, Const3), VarX), VarY))), Add(Add(Multiply(Multiply(Add(Const2,
Multiply(Const3, Const2)), VarY), Multiply(VarY, VarZ)), Multiply(Multiply(Add(Const1,
Multiply(Const3, Add(Const2, Const3))), VarX), VarZ)), Add(Multiply(Multiply(Add(Const1,
Multiply(Const3, Add(Const2, Const3))), VarY), VarZ), Multiply(Multiply(Add(Const2, Multiply(Const3,
Const2)), VarX), Multiply(VarY, VarZ)))))
# test-data/phase2-noite/0403.txt
[2023-12-06 07:49:42.366] INFO: Attempting solution with BFSEnum2Synthesizer
[2023-12-06 07:49:42.714] INFO: Attempting solution with VoltronSynthesizer
[2023-12-06 07:49:42.715] WARNING: No satisfying solution found! Solver returned INFEASIBLE
[2023-12-06 07:49:42.726] WARNING: No satisfying solution found! Solver returned INFEASIBLE
[2023-12-06 07:49:47.729] WARNING: No satisfying solution found! Solver returned UNKNOWN
[2023-12-06 07:49:47.730] WARNING: No satisfying solution found! Solver returned INFEASIBLE
[2023-12-06 07:49:49.117] WARNING: No satisfying solution found! Solver returned INFEASIBLE
[2023-12-06 07:49:54.119] WARNING: No satisfying solution found! Solver returned UNKNOWN
Add(Add(Add(Multiply(Multiply(Multiply(Const3, VarX), VarY), Multiply(Multiply(VarY, VarY), VarY)),
Multiply(Multiply(Multiply(Multiply(Const3, Add(Const1, Multiply(Const3, Const3))), VarX), VarY),
Multiply(VarY, VarY))), Add(Multiply(Multiply(Multiply(Const3, VarX), VarY), Multiply(VarY, VarZ)),
Multiply(Multiply(Multiply(Multiply(Const3, Add(Const2, Multiply(Const3, Const3))), VarX), VarX),
Multiply(VarY, VarY)))), Add(Add(Multiply(Multiply(Multiply(Const3, VarX), VarX),
Multiply(Multiply(VarY, VarY), VarY)), Multiply(Multiply(Multiply(Const3, VarX), VarX),
Multiply(VarX, VarY))), Multiply(Multiply(Multiply(Const3, VarX), VarX), Multiply(VarY, VarZ))))
```

```
# test-data/phase2-noite/0404.txt
[2023-12-06 07:49:55.656] INFO: Attempting solution with BFSEnum2Synthesizer
[2023-12-06 07:49:56.004] INFO: Attempting solution with VoltronSynthesizer
Add(Add(Multiply(Add(Const1, Multiply(Const3, Add(Const2, Const3))), VarZ),
Multiply(Multiply(Add(Const2, Const2), VarX), VarZ)), Multiply(Multiply(Multiply(Const3, Add(Const2,
Const2)), VarY), VarZ))
# test-data/phase2-noite/0405.txt
[2023-12-06 07:49:56.009] INFO: Attempting solution with BFSEnum2Synthesizer
[2023-12-06 07:49:56.365] INFO: Attempting solution with VoltronSynthesizer
Add(Add(Multiply(Add(Const2, Const2), VarY), Multiply(Multiply(Const2, VarX), VarY)), Multiply(VarY,
VarZ))
# test-data/phase2-noite/0406.txt
[2023-12-06 07:49:56.370] INFO: Attempting solution with BFSEnum2Synthesizer
[2023-12-06 07:49:56.717] INFO: Attempting solution with VoltronSynthesizer
Add(Add(Add(Add(Multiply(Const3, Add(Const2, Const2)), Multiply(VarY, VarY)),
Multiply(Multiply(Const2, VarZ), VarZ)), Add(Multiply(Add(Const1, Multiply(Const3, Const2)), VarY),
Multiply(Add(Const2, Multiply(Const3, Const3)), VarZ))), Add(Add(Multiply(Multiply(Const3, VarY),
VarZ), Multiply(Multiply(Add(Const2, Const2), VarX), VarX)), Add(Multiply(Multiply(VarX, VarX),
VarY), Multiply(Multiply(VarX, VarX), VarZ))))
# test-data/phase2-noite/0407.txt
[2023-12-06 07:49:56.749] INFO: Attempting solution with BFSEnum2Synthesizer
[2023-12-06 07:49:57.098] INFO: Attempting solution with VoltronSynthesizer
[2023-12-06 07:49:57.103] WARNING: No satisfying solution found! Solver returned INFEASIBLE
Add(Add(Add(Add(Multiply(Multiply(Add(Const2, Const2), VarX), Multiply(VarY, VarY)),
Multiply(Multiply(Const2, VarX), Multiply(VarZ, VarZ))), Multiply(Multiply(Add(Const2,
Multiply(Const3, Const2)), VarX), VarY)), Add(Add(Multiply(Multiply(Add(Const2, Const2), VarX),
VarZ), Multiply(Multiply(Add(Const3, Const3), VarX), Multiply(VarY, VarZ))),
Multiply(Multiply(Multiply(Const2, VarX), VarX), Multiply(VarY, VarY)))),
Add(Add(Add(Multiply(Multiply(VarX, VarX), Multiply(VarZ, VarZ)), Multiply(Multiply(Add(Const2,
Multiply(Const3, Const2)), VarX), Multiply(VarX, VarY))), Multiply(Multiply(Multiply(Const2, VarX),
VarX), Multiply(VarX, VarY))), Add(Add(Multiply(Multiply(Add(Const2, Const2), VarX), Multiply(VarX,
VarZ)), Multiply(Multiply(VarX, VarX), Multiply(VarX, VarZ))), Multiply(Multiply(Multiply(Const3,
VarX), VarX), Multiply(VarY, VarZ)))))
# test-data/phase2-noite/0408.txt
[2023-12-06 07:49:58.986] INFO: Attempting solution with BFSEnum2Synthesizer
[2023-12-06 07:49:59.338] INFO: Attempting solution with VoltronSynthesizer
Add(Add(Add(Const1, Multiply(Const3, Const3)), Multiply(Add(Const2, Const2), VarY)),
Add(Multiply(Const2, VarZ), Multiply(Multiply(Add(Const3, Const3), VarY), VarZ)))
# test-data/phase2-noite/0409.txt
[2023-12-06 07:49:59.342] INFO: Attempting solution with BFSEnum2Synthesizer
[2023-12-06 07:49:59.690] INFO: Attempting solution with VoltronSynthesizer
[2023-12-06 07:49:59.691] WARNING: No satisfying solution found! Solver returned INFEASIBLE
[2023-12-06 07:49:59.694] WARNING: No satisfying solution found! Solver returned INFEASIBLE
Add(Add(Add(Add(Multiply(Multiply(Multiply(Const2, VarX), VarX), Multiply(VarX, VarX)),
Multiply(Multiply(Const3, Add(Const2, Const2)), VarX)), Multiply(Multiply(Const3, Add(Const2,
Const2)), VarY)), Add(Add(Multiply(Multiply(Add(Const2, Const2), VarX), VarY),
Multiply(Multiply(Const3, VarX), VarZ)), Multiply(Multiply(Const3, VarY), VarZ))),
Add(Add(Add(Multiply(Multiply(VarX, VarY), VarZ), Multiply(Multiply(Add(Const2, Const2), VarX),
VarX)), Multiply(Multiply(Add(Const3, Const3), VarX), Multiply(VarX, VarX))),
Add(Add(Multiply(Multiply(Add(Const3, Const3), VarX), Multiply(VarX, VarY)),
Multiply(Multiply(Multiply(Const2, VarX), VarX), Multiply(VarX, VarY))), Multiply(Multiply(VarX,
VarX), VarZ))))
# test-data/phase2-noite/0500.txt
[2023-12-06 07:50:00.071] INFO: Attempting solution with BFSEnum2Synthesizer
[2023-12-06 07:50:00.431] INFO: Attempting solution with VoltronSynthesizer
[2023-12-06 07:50:00.432] WARNING: No satisfying solution found! Solver returned INFEASIBLE
Add(Add(Add(Multiply(Multiply(Multiply(Const3, Add(Const3, Const3)), VarZ), VarZ),
Multiply(Multiply(Multiply(Const2, VarY), VarY), Multiply(VarZ, VarZ))), Add(VarX,
Multiply(Add(Const2, Const3), VarY))), Add(Add(Multiply(Multiply(Multiply(Const3, Add(Const2,
Const2)), VarY), Multiply(VarZ, VarZ)), Multiply(Multiply(Multiply(Const3, Add(Const2, Const2)),
VarY), Multiply(VarY, VarZ))), Add(Multiply(Multiply(Multiply(Const3, Add(Const3, Const3)), VarY),
VarZ), Multiply(Multiply(Multiply(Const2, VarY), VarY), Multiply(VarY, VarZ)))))

(Terminated manually)
```