A Divide-and-Conquer Approach to Discovering Minimal Realizable Grammars

WILL THOMAS, LOGAN SCHMALZ, and SARAH JOHNSON

1 INTRODUCTION

Our project is based on the work of Semantic-Guided Synthesis (SemGuS) [1]. We modify the algorithm for solving SemGuS problems such that it can discover a minimal realizable grammar. We design and implement two methods for performing this task: (1) a Production Rule Mode and (2) a Non-Terminal Mode. The Production Rule mode exhaustively generates every possible subgrammar by iterating through the elements in the power set of the full grammar's production rules. The Non-Terminal Mode takes a more course-grained approach by iterating only through the elements in the power set of the full grammar's non-terminal symbols. With both approaches, we use a breadth-first search for walking through the subgrammar lattice. We expect that a better, heuristic-based search strategy would vastly improve the performance of our algorithm.

2 BACKGROUND

SemGuS aims to be "a language-agnostic logic-based framework for program synthesis problems over arbitrary semantics". A SemGuS problem is defined by language syntax given as a Regular Tree Grammar (RTG), language semantics given as Constrained Horn Clauses (CHCs), and problem specification given by logical constraints or input-output examples. The feature that sets SemGuS apart from Syntax-Guided Synthesis (SyGuS) is the ability to define a custom semantics for a language, as SyGuS cannot express problems that contain semantics outside of supported theory. SemGuS accepts recursively defined big-step semantics allowing synthesis over imperative programming languages that contain loops with unbounded behaviour. Kim et al. not only designed the SemGuS framework but also developed an algorithm for solving SemGuS problems. The algorithm encodes SemGuS problems as a proof search over CHCs and is capable of both synthesizing programs and proving unrealizability, the latter of which our project relies on.

3 OVERVIEW

Consider a very simple example problem: synthesize a function that returns the first projection of an ordered pair. Our goal is to find a minimal realizable subgrammar for this problem. The full grammar consists of a start nonterminal symbol S with a single production rule E. E is an expression nonterminal symbol with two production rules x and y. x returns the first parameter given to the function while y returns the second. In this example, it is obvious to see that y can be removed from the language and the problem is still realizable.

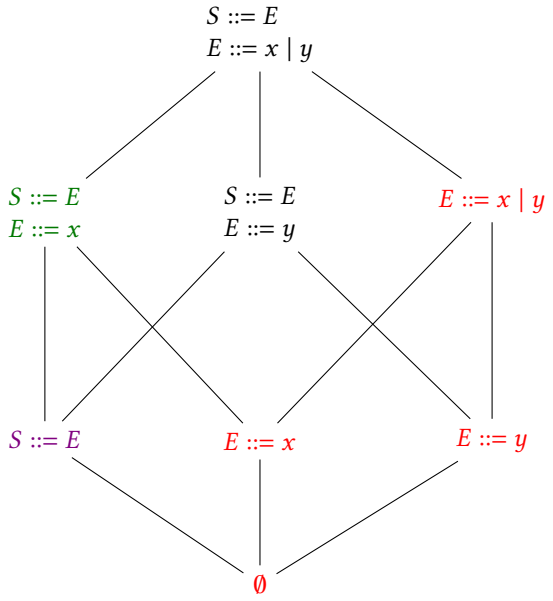

Fig. 1. Subgrammar Lattice

In our work, we design a sound and complete algorithm for finding a minimal realizable subgrammar. Given some grammar, the algorithm generates every subgrammar (including invalid ones) and checks realizability under each one. When using Production Rule Mode, the total set of subgrammars is equal to the power set of the production rules and forms a lattice under the subset relation as shown in Figure 1. The red text denotes invalid grammars due to missing start symbol; the purple text denotes invalid grammars due to missing nonterminal symbol; and the green text denotes the minimal realizable grammar given the behavioural specification.

4 TECHNICAL DETAILS

At its core, our approach had two modes that it could take. We could either remove Non-Terminals from the grammar, or specific production rules related to a Non-Terminal. These approaches, their benefits and limitations will be explored further in the next two subsections. Throughout both examples, we will keep in mind the general purpose grammar presented in Figure 2.

$$\begin{array}{lcccc} \langle G_1 \rangle & ::= & \alpha_{G_{1,1}} & | \ldots | & \alpha_{G_{1,M_1}} \\ \langle G_2 \rangle & ::= & \alpha_{G_{2,1}} & | \ldots | & \alpha_{G_{2,M_2}} \\ \vdots & & \vdots & & \vdots \\ \langle G_N \rangle & ::= & \alpha_{G_{N,1}} & | \ldots | & \alpha_{G_{N,M_N}} \end{array}$$

Fig. 2. Example Technical Grammar G

Additionally, the general purpose algorithm presented in Figure 3 is the approach that will be taken for both the Non-Terminal and the Production Rule modes.

Algorithm 1: Minimal Sub-Grammar

Input: l : List[SpecEvents]
Output: l' : Option[List[SMTCommand]]

```
var declT, defT, hornC, constrs, synthFun = l.filter(classTypeFilter())
var requiredEvs = constrs ::: synthFun
var combinedEvs = defT ::: hornC
for i ← 1 to length(combinedEvs) do
    for part ∈ combinedEvs.combinations(i) do
        var smtEncoding = semgus2SMT(part ::: requiredEvs)
        if checkSat(smtEncoding) then
            return Some(smtEncoding)
        end
    end
end
Otherwise we return None;
```

Fig. 3. General Purpose Realizability Algorithm

The key features of this algorithm are first splitting the problem based upon the type of the "SpecEvents" that are used. In particular, all realizability problems will need to incorporate the constraints and the synthesize function events. Based upon the mode of the algorithm, we will either choose from partitions of the events (the partitions have grouped together the non-terminals with all of their production rules) or choose from all events (for production rule mode). From then, the baseline Messy method for translating a list of spec events into an SMT encoding (as Constrained Horn Clauses) is used. We then check SAT on the output from Messy and return realizable if SAT, or continue if UNSAT.

4.1 Combinatoric Complexity

One important pre-requisite to understanding the design decisions we made for this project is understanding the computational complexity of a combinatoric problem, such as the one we are attempting. While the individual times to compute realizability or unrealizability may be relatively quick, the number of times we have to compute it due to generating grammar combinations is immense. In particular, we must (in the worst case) generate 2^N sub-grammars to remain a decision procedure for realizability.

$$\sum_{i=0}^{N} \binom{N}{x} == 2^N$$

Ultimately, understanding the inherently exponential nature of the problem we must solve will help provide a background on why different modes were created and why optimization may be required for a true implementation to excel.

4.2 Non-Terminal Mode

In this mode, we would remove whole Non-Terminals from the grammar at a time. This is in essence a very coarse grained approach to finding a minimal realizable sub-grammar. Given the example

grammar G from Figure 2, it can be seen that there are N non-terminals in that grammar. Our approach for non-terminal mode will be to start enumerating all combinations of the non-terminals in grammar G. We will start this enumeration from the bottom-up, as the time to process a small grammar and decide either realizability or unrealizability will be minimal. This way if a small grammar is realizable, we can quickly dispatch it to a state-of-the-art synthesis tool for complete synthesis. The downfall of our technique is that by enumerating from the bottom-up of for sub-grammars, if a grammar is only realizable with the majority (or in the worst case all) non-terminals it will take a long time to complete.

One thing that the reader may be considering is that the removal of a non-terminal will violate the "minimal" property of our algorithm. This is technically true, as the output grammar will not contain the least number of both non-terminals and production rules required to complete its synthesis task, but when viewed from the realm of non-terminals it remains true. The non-terminal mode of the algorithm will be guaranteed to yield a sub-grammar with the "minimal" number of non-terminals required for completing the task.

4.3 Production Rule Mode

The general approach in Production Rule mode is quite similar to the Non-Terminal mode described above. The key difference is that instead of grouping together non-terminals and their corresponding production rules, we just remove production rules. Production rule mode can be thought of as a more extreme version of the non-terminal mode: the strengths are improved, but the weaknesses also worsened. If a small realizable sub-grammar exists, Production rule mode is bound to find it quickly and allow for a vast deal of improved synthesis time as the grammar the synthesizer must operate over is quite small. In the case that the problem is not realizable, even with a full grammar, then the time it takes for production rule mode to complete will be 2^K, where $K = \sum_{i=1}^{N} M_i$ (recall N, M for the example grammar in Fig 2). Depending on the size of the grammar, and the amount of production rules required to solve a given synthesis problem, this mode could be highly beneficial or a complete waste of time.

5 IMPLEMENTATION

We initially planned to reimplement Messy from the ground up in OCaml. However, Messy made heavy use of subclasses which we found difficult to work with in OCaml, and our unfamiliarity with the language encourage us to switch to Python. As we got further into the reimplementation using Python, we realized Messy used custom Java libraries that we did not want to reimplement in addition to Messy. In the end, we resigned ourselves to extending the original implementation of Messy in Scala with our technique.

To accomplish this, we generate all possible combinations of grammar SMT constraints (in an appropriate way for either Non-Terminal or Production Rule mode) and generate a new Z3 file for each subgrammar. To facilitate verification of subgrammars, we also implemented a call to Z3 after each subgrammar is generated. If it is realizable (or potentially realizable by the timeout or unknown results), we output the Z3 file and stop testing new subgrammars.

6 EVALUATION

In this section, we will dive into our evaluation, comparing the run time of the original Messy solver to our NT (Non-Terminal) mode. We did not collect data on the running of Prod (Production Rule) mode as the time for each benchmark to complete in Prod mode was extremely long and would establishing a good dataset was infeasible due to time constraints.

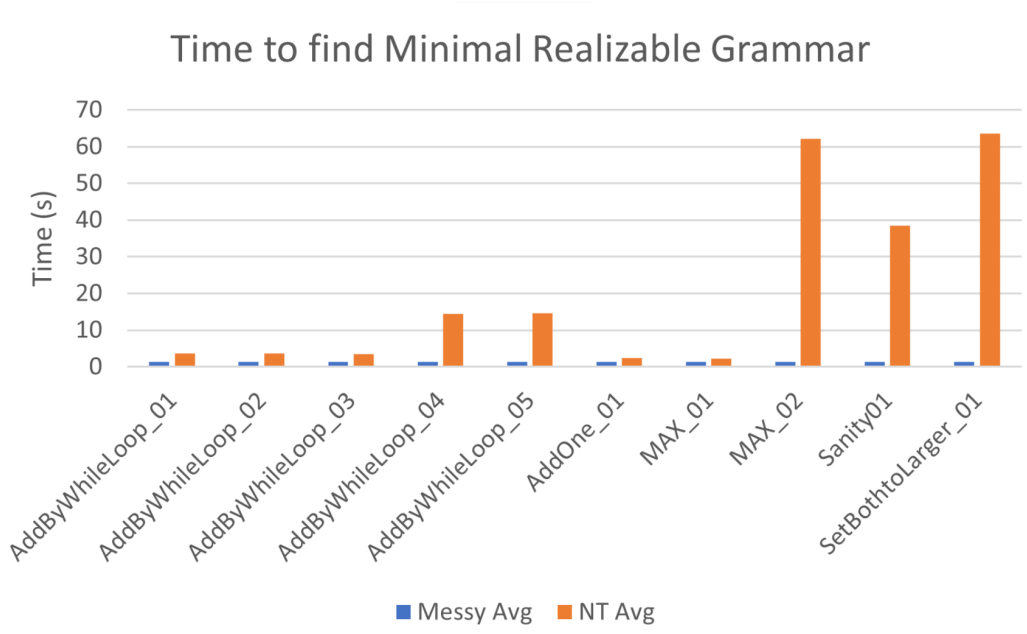

Fig. 4. Benchmarking Results

Unfortunately, both our NT and Prod approaches are slower than the original Messy Solver on all benchmarks. Additionally, our approach was limited in exactly the same ways as the the original Messy solver. For one, it cannot parse SemGuS files that Messy cannot, of which a surprising number of the Messy benchmarks were actually un-parseable. We had to invalidate 8 tests from the 18 "Messy benchmarks" due to the fact that Messy itself could not parse them to even attempt solving.

One great benefit however was our Prod approach was able to discover a minimal sub-grammar that was smaller than the full grammar. For the test AddByWhileLoop_1 (notably the only problem we ran Prod on to completion), we generated a grammar in 434 seconds that used 11 less variables, 2 less relations, and 7 less semantic rules, but was still realizable. This reduction implies that we are able to get rid of certain production rules, but not the non-terminals related to those production rules, since the NT mode was not able to find a minimal sub-grammar for the same test. However, this comes at a very high cost, as the time it took our solver to find the minimal realizable grammar was more than 100× that of the original Messy solver. Further testing would be required to determine if the time saved during synthesis by using this smaller sub-grammar is worth the upfront cost of Prod mode.

7 FUTURE WORK

Firstly, there is some work that remains to be done on the original implementation of Messy, including improving parsing. Adjustments to how Messy does parsing could bring both more flexible representations of SemGuS structures for analysis, as well as provide a better reference implementation for parsing SemGuS problem specifications. This would likely be a primary focus of our future approach.

We believe there are substantial optimizations that can be done to reduce the number of obviously invalid subgrammars that we check. This includes subgrammars without start symbols

and subgrammars which have non-terminals with no productions. The major limitation we faced on this front was that the original Messy internal representation of grammars was not conducive to analysis of the grammar. By improving the internal representation, we could detect invalid grammars and eliminate them from the candidate subgrammars.

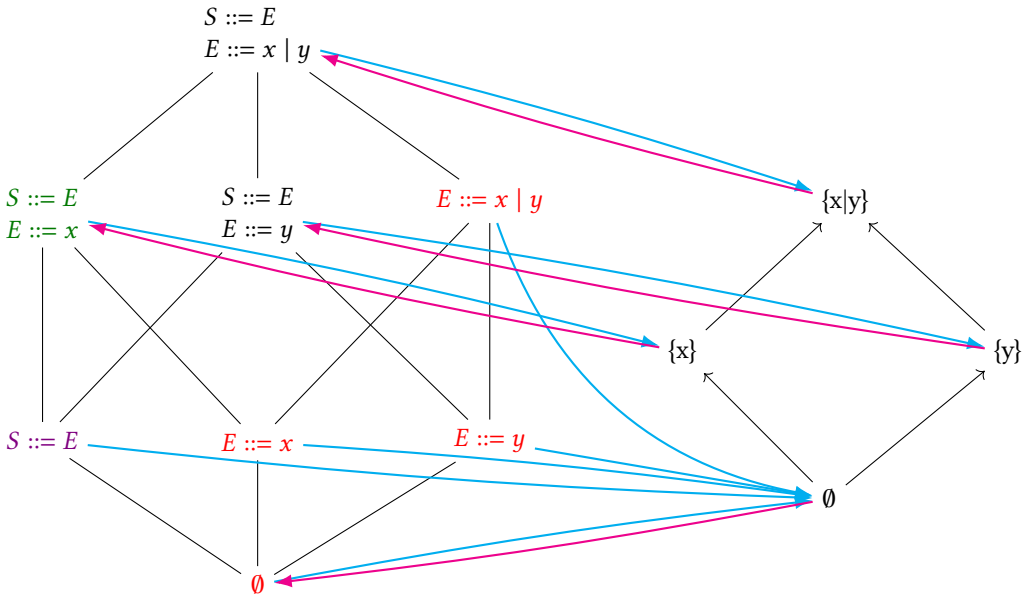

Fig. 5. Galois connection between subgrammars and languages

We additionally believe that a better internal grammar representation could be used to develop methodologies or heuristics for choosing specific subgrammars to maximize learning about the synthesis problem and increase efficiency. In particular, one approach we see potential in is developing a Galois connection between subgrammars and the languages they generate. An example of this idea can be seen in Figure 5.

Finally, the inspiration behind this project was to discover a minimal potentially-realizable grammar and send it to a synthesizer with stronger abilities compared to Messy on the synthesis front. Implementing this feature to send a minimal grammar to a better synthesizer would be worthwhile.

8 CONCLUSION

Although our approach is significantly slower than the original Messy solver algorithm due primarily to the large combinatoric complexity of iterating through subgrammars, our goal of finding minimal realizable grammars was successful on some benchmarks. The efficiency of our algorithm can be improved through several optimizations: (1) discard invalid grammars quickly and (2) develop an effective, heuristic-based search strategy. Grammars missing a start symbol and/or including a nonterminal symbol with no production rules are considered invalid and should be discarded immediately. We were unable to implement this in our project due to the unweildy representation of grammars in Messy. We believe that a Galois connection may be the answer to developing an improved search strategy but did not have the time to complete our research on this approach.

REFERENCES

[1] Jinwoo Kim, Qinheping Hu, Loris D'Antoni, and Thomas Reps. 2021. Semantics-Guided Synthesis. *Proc. ACM Program. Lang.* 5, POPL, Article 30 (jan 2021), 32 pages. https://doi.org/10.1145/3434311