Supplement: Selecting Representative Examples for Program Synthesis

Yewen Pu [1] **Zachery Miranda** [1] **Armando Solar-Lezama** [1] **Leslie Pack Kaelbling** [1]

Checking Representativeness

The following algorithm verifies whether a subset D' is representative of D, i.e. $\forall s \in SD'(s) \Rightarrow D(s)$

Algorithm 1 Is Representative

Input: data D, subset D'
Output: boolean
for $d \in D \setminus D'$ **do**
  if synthesize($D' \cup \{\neg d\}$ **then**
    return false
  end if
end for
return true

Proof: For simplicity let us consider D as a set of constraints $\{d_1, d_2, ..., d_n\}$. If the synthesizer can find some s that satisfies $[D' \wedge \neg d_i](s)$, then we have found a solution $s \in S$ which is consistent with D' but contradicts $d_i \in D$. Therefore D' is **not** a *representative subset* of D. If no solution can be found for any $[D' \wedge \neg d_i](s)$, then every d_i is consistent with D' which means that $D'(s) \Rightarrow D(s)$ $\square$.

Selection Heuristic for DFA Synthesis

We constructed a sample heuristic function for DFAs using the power of suffixes in DFAs. Suffixes are powerful because they are the reverse path from potentially the accept state of the DFA if the value is true. The heuristic's process is shown on an example in Figure 1. We realized that the suffix of a string has a lot to do with whether a string was accepted so first we reverse all of the strings. Then we create a binary search tree using the values in the strings. In each node, we keep track of how many children examples (including that node) have each corresponding truth value. For selection, you start at the root and look at every node's truth values to see if all children of that node are all true or all false. If the truth is absolute, then you take a random child of the node and add it as an example, otherwise you analyze both children of that node. Once all nodes are analyzed that need to be analyzed, then the resulting examples are returned as the subset.

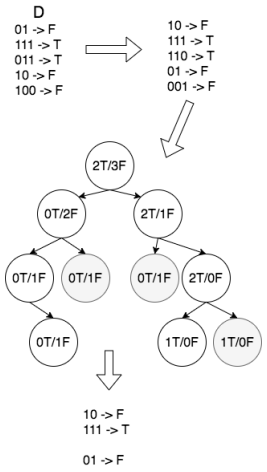

Figure 1. The process of our DFA selection heuristic on a sample of data.

Synthesized Drawing Programs

Figure 2 shows some synthesized drawing programs. Each cell consists of: The target rendering, the subset of selected examples, the neural-network estimation of the rendering, and the synthesized parameters for the $draw$ function.

Neural Network Architectures

Order Synthesis Figure 3 shows the architecture used for the ordering domain, as the domain is small $n = 10$ there can only be as many as 100 pair-wise relationships. Therefore, we encode the neural network without any neighborhood structure but takes in the entire subset D' at once. Each i, j element in the input denotes whether $i < j$ or $i \geq j$ or that it is undefined in D'. The new query is two 1-hot encoded vectors asking if $x < y$ conditioned on the subset D'. This architecture was trained on 1000000 randomly generated $(D', x < y)$ pairs from randomly sampled permutations. Learning rate of 0.0001 with AdamOptimizer.

*Equal contribution [1]Massachusetts Institute of Technology. Correspondence to: Yewen Pu <yewenpu@mit.edu>.

Proceedings of the 35^{th} International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018.

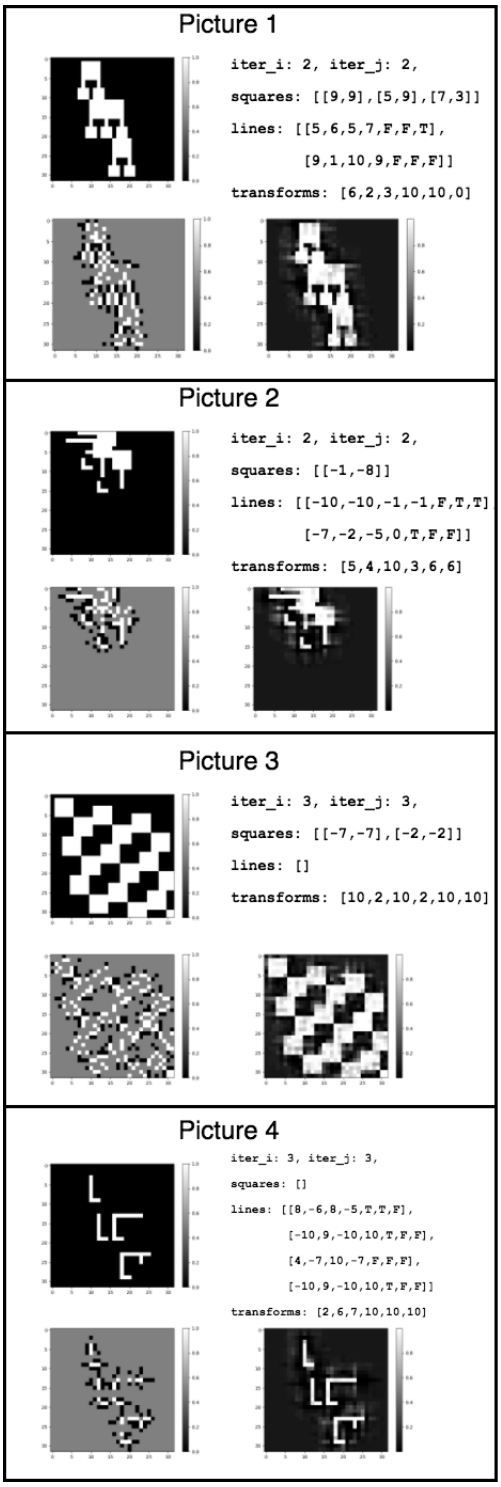

Figure 2. Sample pictures generated using the drawing DSL.

DFA Synthesis Figure 4 shows the architecture used for the DFA synthesis task. The top-10 closest strings in terms of prefix and suffix, along with whether that string was accept / rejected was encoded into a hidden representation. A new string is concatenated with this hidden representation to output whether that new string should be accepted or rejected. This architecture was trained on 100000 randomly generated $D', newstr, accept_{newstr}$ pairs from randomly sampled strings on a randomly sampled DFA. Learning rate of 0.0001 with AdamOptimizer.

Programmatic Drawing Synthesis Figure 5 shows the architecture used for the drawing synthesis task. We simply use a 7×7 convolutional neural network with 20 hidden units. This architecture was trained on 20000 randomly sampled renderings. Learning rate of 0.001 with AdamOptimizer.

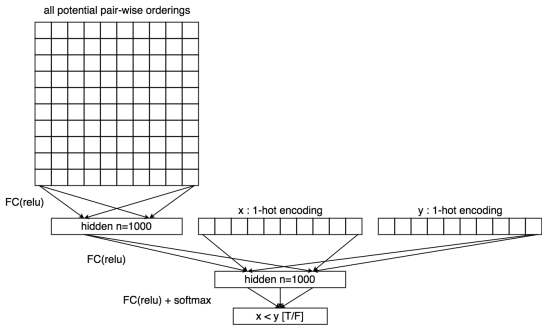

Figure 3. The neural network architecture for the ordering problem.

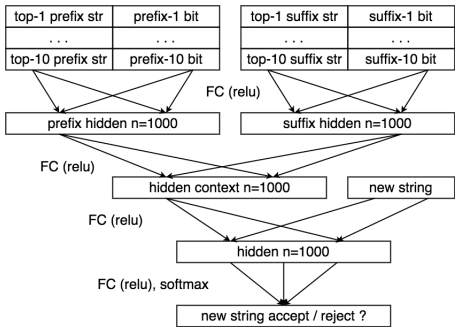

Figure 4. The neural network architecture for the DFA problem.

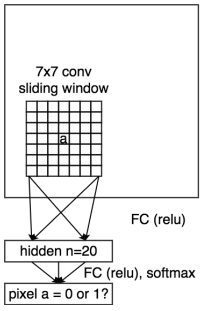

Figure 5. The neural network architecture for the drawing problem.