Program Synthesis for Complex Software Systems

Yanju Chen
Computer Science Department
University of California, Santa Barbara
05/03/2022
Overview

01 Complex Software Systems
Complex Software Systems Around Us

02 Program Synthesis for Modern Web Browsers
Tree Traversal Synthesis Using Domain-Specific Symbolic Compilation[1]

03 Program Synthesis for Deep Learning Systems
Visualization Question Answering Using Introspective Program Synthesis[2]

04 Conclusions and Proposals
Deduction-Powered Neural Program Synthesis: A Multi-Modal Perspective

Complex Software Systems

Complex Software Systems Around Us

• Complex Systems
• Complex Software Systems Around Us
  • Operating Systems
  • Modern Web Browsers
  • Deep Learning Systems
Complex Systems

organisms
cognition
weather system
Complex Software Systems Around Us

Operating Systems
- e.g., Serval\[^1\]

Modern Web Browsers
- e.g., Cassius\[^2\]

Deep Learning Systems
- e.g., LIME\[^3\]

Program Synthesis for Modern Web Browsers

Tree Traversal Synthesis Using Domain-Specific Symbolic Compilation

- Motivations
  - Modern Web Browsers
  - Tree Traversals
  - A Motivating Example
- Existing Approaches & Challenges
- Overview: HECATE
- Attribute Grammar & Traversal Language
- General-Purpose Symbolic Compilation
- Domain-Specific Symbolic Compilation
- Complexity Analysis
- Evaluation
  - GRAFTER
  - A Case Study: RenderTree
  - FTL
- Session Conclusions

Modern Web Browsers

- Motivations
  - Multiple Modules
  - Natural Language Specifications (W3C)
  - Fault Tolerance
  - Legacy Codebase
  - ...

- Program Synthesis for Modern Web Browsers
Tree traversals are widely used and play important roles.
- Motivations -

A Motivating Example

• Synthesizing A Toy Layout Engine
  • Two classes, Four Attributes
  • Attribute Grammar

dependencies in n₁’s attributes

class definitions

symbolic traversal

dependencies in n₁’s attributes

concrete traversal

symbolic traversal

Attribute Grammar

```
1 interface Box{
  2   input w0, h0: int;
  3   output w1, w1, h1, h1: int;
  4 }

5 class Inner: Box{
  6   children {
  7     nx: Optional[Box];
  8     fc: Optional[Box];
  9 }
 10 rules {
 11     self.w := max( self.w0, fc.w1 );
 12     self.w1 := max( self.w1, nx.w1 );
 13     self.h := max( self.h0, fc.h1 );
 14     self.h1 := self.h + nx.h1;
 15 }
}

18 class Leaf: Box{
  19   children {
 20 }
 21 rules {
 22     self.w := self.w0;
 23     self.w1 := max( self.w, nx.w1 );
 24     self.h := self.h0;
 25     self.h1 := self.h + nx.h1;
 26 }
}
```
Existing Approaches & Challenges

- Automata Based: TreeFuser[1] and GRAFTER[2]
  - Deterministic Rewrite Rules (Complex to Maintain)

- Synthesis Based: FTL[3]
  - Constraints Generated by Domain Experts (Manual and Error-Prone)

- General-Purpose Symbolic Compilation
  - Solver-Aided Programming Languages, e.g., Rosette[4]
  - Path Explosions & Complex Constraint System

Overview: HECATE

- A CEGIS Framework for Tree Traversal Synthesis
- A Domain-Specific Trace Language
  - For Disentangling Complex Dependencies in Trees
  - For Generating Easy-to-Solve Constraints for Tree Traversal Synthesis
- A Tool Called HECATE
  - For Expressive, Efficient and Flexible Tree Traversal Synthesis
- Synthesis Using HECATE -

Attribute Grammar & Traversal Language

\[
\begin{align*}
\langle \text{interface} \rangle &::= \text{interface} \langle \text{id} \rangle \left\{ \langle \langle \text{tup} \rangle \rangle^* \right\} \\
\langle \text{class} \rangle &::= \text{class} \langle \text{tup} \rangle \left\{ \langle \text{children} \rangle \langle \text{rules} \rangle \right\} \\
\langle \text{children} \rangle &::= \text{children} \left\{ \langle \langle \text{tup} \rangle \rangle^* \right\} \\
\langle \text{rules} \rangle &::= \text{rules} \left\{ \langle \langle \text{cstmt} \rangle \rangle^* \right\} \\
\langle \text{tup} \rangle &::= \langle \text{id} \rangle \langle \langle \text{id} \rangle \rangle \langle \langle \text{id} \rangle \rangle^* \\
\langle \text{sel} \rangle &::= \langle \langle \text{id} \rangle \rangle \langle \langle \text{id} \rangle \rangle \langle \langle \text{id} \rangle \rangle \\
\langle \text{expr} \rangle &::= \langle \text{const} \rangle | \langle \text{sel} \rangle | f( \langle \text{expr} \rangle^* ) \\
& \quad | \langle \text{expr} \rangle \langle \text{op} \rangle \langle \text{expr} \rangle | \text{fold}( \langle \text{expr} \rangle^* ) \\
& \quad | \text{if}( \langle \text{expr} \rangle ) \text{then} \langle \text{expr} \rangle \text{else} \langle \text{expr} \rangle \\
\langle \text{cstmt} \rangle &::= \langle \text{sel} \rangle := \langle \text{expr} \rangle \\
\langle \text{op} \rangle &::= + | - | \times | \div | ... \\
\end{align*}
\]

\( f \in \text{functions} \quad \langle \text{const} \rangle \in \text{constants} \quad \langle \text{id} \rangle \in \text{identifiers} \)

Figure 6: Syntax for attribute grammar \( \mathcal{L}_a \).

\[
\begin{align*}
\langle \text{traversal} \rangle &::= \text{traversal} \langle \text{id} \rangle \left\{ \langle \text{case} \rangle^* \right\} \\
\langle \text{case} \rangle &::= \text{case} \langle \text{id} \rangle \left\{ \langle \langle \text{tstmt} \rangle \rangle^* \right\} \\
\langle \text{recur} \rangle &::= \text{recur} \langle \text{node} \rangle \\
\langle \text{iterate} \rangle &::= \text{iterate} \left\{ \langle \langle \text{tstmt} \rangle \rangle^* \right\} \\
\langle \text{parallel} \rangle &::= \text{parallel} \left\{ \langle \langle \text{tstmt} \rangle \rangle^* \right\} \\
\langle \text{eval} \rangle &::= \text{eval} \langle \text{cstmt} \rangle \\
\langle \text{tstmt} \rangle &::= i | \langle \text{recur} \rangle | \langle \text{iterate} \rangle | \langle \text{eval} \rangle \\
\langle \text{id} \rangle \in \text{identifiers} \quad \langle \text{node} \rangle \in \text{nodes} \\
\end{align*}
\]

Figure 7: Syntax for tree traversal language \( \mathcal{L}_t \).

* Please refer to the paper for more details.
- Synthesis Using HECATE

General-Purpose Symbolic Compilation

- Constraint System
  - Semantic Constraints
    
    \[
    (\sigma(\text{none}, t_2) \implies \text{true})
    \]

    \[
    \forall (\sigma(\text{Inner}.w1, t_2) \implies \delta(\zeta(n_1, \text{self.w}), t) \land \delta(\zeta(n_1, \text{nx.w1}), t)
    \land \neg \delta(\zeta(n_1, \text{self.w1}), t))
    \]

    \[
    \forall (\sigma(\text{Inner}.w, t_2) \implies \delta(\zeta(n_1, \text{self.w0}), t) \land \delta(\zeta(n_1, \text{fc.w1}), t)
    \land \neg \delta(\zeta(n_1, \text{self.w}), t))
    \]

    \[
    \forall (\sigma(\text{Inner}.h1, t_2) \implies \delta(\zeta(n_1, \text{self.h}), t) \land \delta(\zeta(n_1, \text{nx.h1}), t)
    \land \neg \delta(\zeta(n_1, \text{self.h1}), t))
    \]

    \[
    \forall (\sigma(\text{Inner}.h, t_2) \implies \delta(\zeta(n_1, \text{self.h0}), t) \land \delta(\zeta(n_1, \text{fc.h1}), t)
    \land \neg \delta(\zeta(n_1, \text{self.h}), t))
    \]

  - Auxiliary Constraints

    - Every slot should be filled with at most one rule.
    - Every rule should be used by only one slot.

- Every slot should be filled with at most one rule.
- Every rule should be used by only one slot.

"choose one to schedule"
"All dependencies should have been ready"
"Target attribute has not been scheduled"

Number of timesteps grows as example trees become larger, which increases the complexity.
- Synthesis Using HECATE

**Domain-Specific Symbolic Compilation**

- **[Traversals]** Given a tree, a traversal defines a total order relation $\prec$ over the set of all locations of the tree.

- **[Example]** A concrete post-order traversal on the example tree yields the following total order of locations:

  \[
  n_4.w < n_4.h < n_4.w1 < n_4.h1 < n_3.w < n_3.h < n_3.w1 < n_3.h1 \\
  < n_1.w < n_1.h < n_1.w1 < n_1.h1 < n_2.w < n_2.h < n_2.w1 < n_2.h1 \\
  < n_0.w < n_0.h < n_0.w1 < n_0.h1
  \]

We can map a traversal from time domain to relational domain.

Such a traversal can be both concrete or symbolic.
- Synthesis Using HECATE

Domain-Specific Symbolic Compilation

• A Symbolic Trace Language

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(choose ([a_1, \ldots a_n]))</td>
<td>choose one from the attributes</td>
</tr>
<tr>
<td>(alloc)</td>
<td>returns a fresh concrete location</td>
</tr>
<tr>
<td>(read (n.a))</td>
<td>logs a read from (n.a)</td>
</tr>
<tr>
<td>(write (n.a))</td>
<td>logs a write to (n.a)</td>
</tr>
</tbody>
</table>

\[(\text{assume } \sigma(\text{Inner.h, } t_2) \quad \text{(read } n_1.h_0) \quad \text{(read } n_3.h_1) \quad \text{(write } n_1.h)\)]

• [0-1 Integer Linear Programming] Given coefficients \(a, b\) and \(c\), the 0-1 ILP problem is to solve for \(x\) as follows:

\[
\min \sum_i c_i x_i \quad s.t. \forall i, j. \sum_j a_{i,j} x_j \leq b_i,
\]

where all entries are integers and in particular \(x_j \in \{0,1\}\).
Domain-Specific Symbolic Compilation

(assume $\sigma(\text{Inner.h}, t_2)$
(\text{read } n_1.h0) (\text{read } n_3.h1) (\text{write } n_1.h))

- Constraint System
  - Dependency Constraints
    \[
    \sigma(\text{Inner.h}, t_2) \leq \sum_{t_0 \leq t} \kappa[n_1.h0, t_0]
    = \sigma(\text{Inner.h0}, t_0) + \sigma(\text{Inner.h0}, t_1), \quad \text{(read for } n_1.h0)
    \]
    \[
    \sigma(\text{Inner.h}, t_2) \leq \sum_{t_0 \leq t} \kappa[n_3.h1, t_0]
    = \sigma(\text{Leaf.h1}, t_4) + \sigma(\text{Leaf.h1}, t_5)
    + \sigma(\text{Leaf.h1}, t_6) + \sigma(\text{Leaf.h1}, t_7), \quad \text{(read for } n_3.h1)
    \]
  - Validity Constraints
    \[
    \forall t. \sum_a \sigma(a, i) \leq 1, \quad - \text{Every slot should be filled with at most one rule.}
    \]
    \[
    \forall a. \sum_i \sigma(a, i) = 1, \quad - \text{Every rule should be used by only one slot.}
    \]

"$n_1.h0$ should have been scheduled somewhere before the current corresponding location"

Constraints are not talking about $t$ anymore, but about domain-specific relations now.
Complexity Analysis

- Synthesis Using HECATE

Start the chain (assume \( \sigma(\text{Inner} \cdot h_{t1}) \))

\( \text{read } R_{1}, h_{0} \) (read \( R_{2}, h_{1} \)) (write \( R_{3}, h_{1} \))

Time 12.5

...
Evaluation

• Research Questions

  • RQ1. Performance: What is the performance of synthesized traversals, compared to those generated by state-of-the-art traversal synthesizers?

  • RQ2. Expressiveness: Is HECATE's tree language expressive enough? In particular, can it express prevailing tree traversal synthesis problems and solve them?

  • RQ3. Flexibility: Can HECATE be extended to explore traversals of different design choices?

  • RQ4. Efficiency: What is the benefit of the domain-specific encoding compared to general-purpose encoding?
Comparison against Grafter\textsuperscript{[1]}

- Grafter
  - Static Dependence Analysis
  - Access Automata
- Benchmarks (Adapted from Grafter)
  - Five Real-World Representative Problem
    - Binary Search Tree
    - Fast Multipole Method
    - Piecewise Functions
    - Abstract Syntax Tree
    - Layout Rendering Tree

Table 2: Comparison between Grafter, Hecate and Hecate\textsuperscript{G}
(with general-purpose encoding). The table shows total synthesis time (synthesis + verification) in second.

<table>
<thead>
<tr>
<th>Benchmark</th>
<th># of Rules</th>
<th>Grafter</th>
<th>Hecate</th>
<th>Hecate\textsuperscript{G}</th>
</tr>
</thead>
<tbody>
<tr>
<td>BinaryTree</td>
<td>16</td>
<td>2.6</td>
<td>1.1</td>
<td>3.2</td>
</tr>
<tr>
<td>FMM</td>
<td>14</td>
<td>7.6</td>
<td>1.0</td>
<td>1.6</td>
</tr>
<tr>
<td>Piecewise</td>
<td>12</td>
<td>12.6</td>
<td>2.1</td>
<td>3.1</td>
</tr>
<tr>
<td>AST</td>
<td>136</td>
<td>151.7</td>
<td>20.6</td>
<td>73.4</td>
</tr>
<tr>
<td>RenderTree</td>
<td>50</td>
<td>62.0</td>
<td>4.1</td>
<td>10.1</td>
</tr>
</tbody>
</table>

A Case Study: RenderTree

- Evaluation

• A Total of Five Rendering Passes
  1. Resolving Flexible Widths
  2. Resolving Relative Widths
  3. Computing Heights
  4. Propagating Font Styles
  5. Finalizing Element Positions

• Variants of Different Synthesizers
  • **GRAFTER**
  • **HECATE**⁺ : Sequential, Linked List
  • **HECATE**𝑉 : Sequential, Vector
  • **HECATE**𝑷 : Parallel, Vector

Figure 11: Running time of fused traversals compared to the unfused baseline.

With minimal efforts, Hecate can effectively explore traversals of different design choices.
Synthesizing Layout Engine in FTL\textsuperscript{[1]}

- FTL
  - Specialized for Layout Engine
  - Prolog Style Declarative Language for Partial Schedules
- Benchmarks (Adapted from FTL)
  - CSS\textemdash float
  - CSS\textemdash margin
  - CSS\textemdash full

\begin{table}[h]
\centering
\begin{tabular}{|c|c|}
\hline
Name       & # of Rules \\
\hline
CSS\textemdash float & 192 \\
CSS\textemdash margin & 178 \\
CSS\textemdash full & 244 \\
\hline
\end{tabular}
\end{table}

\textbf{Figure 15: Comparison against FTL: benchmark statistics (left) and results (right).}

Session Conclusions

- HECATE: A Novel Framework for Tree Traversal Synthesis
- Domain-Specific Symbolic Compilation
- Performance, Expressiveness, Flexibility and Efficiency

Hecate is open-sourced and publicly available.

https://github.com/chyanju/Hecate
03

Program Synthesis for

Deep Learning Systems

Visualization Question Answering Using Introspective Program Synthesis\[1\]

- Deep Learning Systems
- VQA: A Motivating Example
- Existing Approaches & Challenges
- Observations
- Overview: POE
- A Walkthrough of POE
- System Workflow in POE
- Introspective Program Synthesis
- Abstract Program Synthesis with Noisy Specification
- Optimal Program Synthesis for Explanation Refinement
- Evaluation
  - Performance
  - Ablation Study
  - Effectiveness
  - Explainability
- Discussions & Session Conclusions

\[1\] Visualization Question Answering Using Introspective Program Synthesis. Yanju Chen, Xifeng Yan, Yu Feng. PLDI 2022.
- Motivations -

Deep Learning Systems

- Data-Driven
- "Black Box"
- Deep & Large-Scale
- ...
VQA: A Motivating Example
(Visualization Question Answering)

- Motivations

**Given a stacked bar chart that represents opinions for future economic growth for different countries, a user describes her query based on the visualization in natural language:**

Which *country's* economy will get *most worse* over next 12 months?

- **A Visualization Question Answering (VQA) task is to design an algorithm that automatically finds the answer to a natural language query based on a given visualization.**
Existing Approaches & Challenges

- **Motivations**

- **Pipelined System: VisQA** [1]
  - The first one that specializes for VQA tasks
  - Errors propagate between different system modules

- **Fully Supervised Machine Learning: SmBoP** [2], **NL2code** [3]
  - Requires manual annotated logic forms / programs as supervised training data
  - Targeted at Table Question Answering (TQA) / code generation tasks

- **Weakly Supervised Machine Learning: TaPas** [4]
  - Requires only question-answer pairs for training – easy to collect corpus
  - Targeted at TQA tasks

---

Observations

- Motivations

• For mainstream weakly supervised approaches that directly output VQA answers, they are:
  • non-trivial for human beings to understand, and
  • hard to fix if there's error in model reasoning/answer.

• Can we synthesize the hidden reasoning procedures to explain the model's predictions? So that we can:
  • help human beings understand the model behavior, and
  • fix potential model reasoning issues.

• In this work, we investigate such a new problem setting where:
  • not all the predictions are correct, and
  • predictions may conflict with each other.

Figure 4. Example tables showing how one can derive similar programs to get conflicting outputs.
Overview: POE

• Fixing Deep Learning Model's (Noisy) Outputs via **Introspective Program Synthesis**
  • Search Space Induction via **Abstract Program Synthesis**
  • Finding Best Consistent Programs via **Optimal Program Synthesis**

In the context of this work, we use *programs* and *explanations* interchangeably.
A Walkthrough of POE -

**Visualization**

<table>
<thead>
<tr>
<th>Country</th>
<th>opinion</th>
<th>% color</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brazil</td>
<td>Improve</td>
<td>84 blue</td>
</tr>
<tr>
<td>Brazil</td>
<td>Remain the same</td>
<td>12 orange</td>
</tr>
<tr>
<td>Brazil</td>
<td>Worsen</td>
<td>5 red</td>
</tr>
<tr>
<td>China</td>
<td>Improve</td>
<td>33 blue</td>
</tr>
<tr>
<td>China</td>
<td>Remain the same</td>
<td>9 orange</td>
</tr>
<tr>
<td>China</td>
<td>Worsen</td>
<td>2 red</td>
</tr>
<tr>
<td>Tunisia</td>
<td>Improve</td>
<td>75 blue</td>
</tr>
<tr>
<td>Greece</td>
<td>Improve</td>
<td>60 red</td>
</tr>
<tr>
<td>Greece</td>
<td>Remain the same</td>
<td>10 orange</td>
</tr>
<tr>
<td>Greece</td>
<td>Worsen</td>
<td>81 red</td>
</tr>
</tbody>
</table>

**Data**

Which country's economy will get *most worse* over next 12 months?

**Explanation#1**

- $T_0 = \text{pivot}(T, \text{"opinion"}, \text{"%"})$
- $T_1 = \text{select}(T_0, \text{"Improve"})$ eqmax, null
- $T_2 = \text{project}(T_1, \text{["Country"]})$

**Explanation#2**

- $T_0 = \text{pivot}(T, \text{"opinion"}, \text{"%"})$
- $T_1 = \text{select}(T_0, \text{"Worsen"})$ eqmax, null
- $T_2 = \text{project}(T_1, \text{["Country"]})$

**Query**

- Which country's economy will get *most worst* over next 12 months?

---

**Figure 2.** A motivating example on data of opinions for future economic growth for different countries. Left: A visualization of stacked bar chart for illustrating the data distribution; Middle: The corresponding table format of the data; Right: Example checking semantic consistency between three parties: data, query and explanation. Explanation#1 doesn’t fit since no keyword in the query shares similar meaning with *Improve* in the data and *Improve* in the explanation; Explanation#2 satisfies semantic consistency.
A Walkthrough of POE

- Original TAPAS Outputs:

(0.78, Brazil), (0.67, Japan), (0.55, Greece), ...

- POE's Abstract Program Synthesis Outputs:

```python
1 project(select(pivot(T, φ, φ), φ, φ, φ, φ)
2 project(select(T, φ, φ, φ, φ)
3 ...
```

- POE's Optimal Program Synthesis Outputs:

```python
project(select(pivot(
    T, "opinion", ",\%
), "Improve", eqmax, null), ["Country"])

project(select(pivot(
    T, "opinion", ",\%
), "Worsen", eqmax, null), ["Country"])
```

Figure 3. Syntax of a toy DSL for data wrangling.
System Workflow in POE

- Synthesis Using POE -

Deep Learning Model → Question → Visualization → DSL

Answer 1 → Abstract Search Spaces → Optimal Synthesis → Interpreter → Optimal Answer

Answer 2

... → Answer k
Introspective Program Synthesis

Given:

1. a visualization question answering task $\mathcal{T} = (I, Q)$ where $I$ is the visualization and $Q$ is the question in natural language,
2. a domain-specific language $L = (V, \Sigma, R, S)$ and
3. a weakly supervised deep learning model $\pi$ that predicts top-$k$ answers $\mathcal{A} = \pi(I, Q)$,

the goal of introspective program synthesis is to find a complete program $P$ such that $S \Rightarrow P$ and $P$ optimizes the following objectives $\mathcal{O}$:

$$
P^* = \arg\max_P J_{\mathcal{T}, \pi}(P)
$$

$$
= \arg\max_P \sum_{o \in \mathcal{O}} \theta_o \cdot o(I, Q, \mathcal{A}, P),
$$

where $P^*$ is the optimal program, $J$ is a cumulative term of weighted objectives $o \in \mathcal{O}$.
Abstract Program Synthesis with Noisy Specification

- Synthesis Using POE -

• Intuition: Narrow down program search space to such a sweet spot that:
  • respects the model outputs, and
  • promote synthesis efficiency.

Figure 6. Different granularities that affect the algorithm search space. An input-output pair is denoted by a triangle.


Abstract Synthesis Breakdown:

- feasible for all examples
- feasible for all examples
- feasible for "Brazil", "Japan", "China"
- feasible for "Brazil", "Japan"
Optimal Program Synthesis for Explanation Refinement

- Intuition: Maximize consistency between explanation, visualization and query.

- Hard Constraints
  - There is exactly one terminal that maps to a hole.
  - There is exactly one abstract program that is chosen.
  - Each hole belongs to exactly one abstract program.
  - Each visualization unit (cell value) can map to at most one linguistic unit in the query.
  - The above constraints apply to holes within the same abstract programs.

```
Data
<table>
<thead>
<tr>
<th>Country</th>
<th>opinion</th>
<th>% color</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brazil</td>
<td>Improve</td>
<td>64 blue</td>
</tr>
<tr>
<td>Brazil</td>
<td>Remain the same</td>
<td>12 orange</td>
</tr>
<tr>
<td>Brazil</td>
<td>Worsen</td>
<td>5 red</td>
</tr>
<tr>
<td>China</td>
<td>Improve</td>
<td>83 blue</td>
</tr>
<tr>
<td>China</td>
<td>Remain the same</td>
<td>9 orange</td>
</tr>
<tr>
<td>China</td>
<td>Worsen</td>
<td>2 red</td>
</tr>
<tr>
<td>Tunisia</td>
<td>Improve</td>
<td>76 blue</td>
</tr>
</tbody>
</table>

T0 = pivot(T, "opinion", "%")
T1 = select(T0, "Improve", eqmax, null)
T2 = project(T1, [Country])

Explanation#1
```

```
Query
Which country's economy will get worst over next 12 months?
```

```
Explanation#2
T0 = pivot(T, "opinion", "%")
T1 = select(T0, "Worsen", eqmax, null)
T2 = project(T1, [Country])
```

```
project(select(pivot(T, φ₀, φ₁), φ₂, φ₃, φ₄, φ₅))
```
Optimal Program Synthesis for Explanation Refinement

- **NSYN**: Near-Synonym Linguistic Engine
  - A linguistic engine that determines whether two linguistic units are near-synonyms (semantically similar)

  \[ \text{NSYN}(\text{“high”, “highest”}) > \text{NSYN}(\text{“high”, “low”}) \]

- Soft Constraints / Objective Function

\[ \sum_{w \in V_w} \sum_{t \in V_t} (1 - \text{NSYN}(w, t)) \cdot x_w^t + \sum_{p \in V_P} \text{PPL}(P) \cdot u^p \]

Two units mapped should be as similar as possible

More common abstract programs are preferred.
Evaluation

• Research Questions
  • **RQ1. Performance**: How does POE compare against state-of-the-art tools on visualization queries?
  • **RQ2. Effectiveness**: Can POE rectify wrong answers proposed by other tools?
  • **RQ3. Explainability**: Does POE synthesize explanations that well capture the question intentions and make sense to human end-users?
  • **RQ4. Ablation**: How significant is the benefit of abstract synthesis and optimal alignment?

• Benchmarks
  • **629 Visualization Question Answering Tasks from VisQA**[1]
    • Real-World Data Sources
    • Non-Trivial Questions from Real Users
    • Wide Coverage of Question Types

---

- **Performance**

  • Comparison against TAPOAS\(^1\) and VisQA\(^2\)
    - VisQA: +8%
    - POE (top-1): +23%
    - POE (top-3): +27%
    - POE (top-5): +28%

  • Stats of Different Questions Types
    - Retrieval
    - Comparison
    - Aggregation
    - Other
    - Total

**Table 1.** Comparison on number of benchmarks solved by different tools across different types of questions.

<table>
<thead>
<tr>
<th>question type</th>
<th>total</th>
<th>VisQA (baseline)</th>
<th>TAPOAS</th>
<th>POE (top-1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>retrieval</td>
<td>183</td>
<td>101 (55%)</td>
<td>98 (54%)</td>
<td>123 (67%)</td>
</tr>
<tr>
<td>comparison</td>
<td>87</td>
<td>50 (57%)</td>
<td>0 (0%)</td>
<td>71 (82%)</td>
</tr>
<tr>
<td>aggregation</td>
<td>253</td>
<td>92 (36%)</td>
<td>119 (47%)</td>
<td>161 (64%)</td>
</tr>
<tr>
<td>other</td>
<td>106</td>
<td>31 (29%)</td>
<td>12 (11%)</td>
<td>15 (14%)</td>
</tr>
<tr>
<td>total</td>
<td>629</td>
<td>274 (44%)</td>
<td>229 (36%)</td>
<td>370 (59%)</td>
</tr>
</tbody>
</table>

**Figure 7.** Performance comparison between the original pipeline from VisQA (baseline), TAPOAS and POE.

POE can greatly boost performance of weakly supervised models.

POE is effective across different types of benchmarks.

\(^{1}\) TaPOAS: Weakly Supervised Table Parsing via Pre-training. Herzig, J. et al. ACL 2020.

- Evaluation -

Ablation Study

- Variants of POE
  - $\text{POE}^O$: only performs optimal synthesis on the full search space
  - $\text{POE}^A$: only performs abstract synthesis followed by an enumerative search to pick the first concrete program

Table 2. Comparison between TAPAs and different ablated variants of POE.

<table>
<thead>
<tr>
<th>variant</th>
<th>TAPAS</th>
<th>POE</th>
<th>POE$^A$</th>
<th>POE$^O$</th>
</tr>
</thead>
<tbody>
<tr>
<td>solved</td>
<td>229 (+0%)</td>
<td>370 (+23%)</td>
<td>194 (-5%)</td>
<td>357 (+21%)</td>
</tr>
<tr>
<td>delta (%)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#timeout</td>
<td>-</td>
<td>36</td>
<td>586</td>
<td>58</td>
</tr>
</tbody>
</table>

Both procedures are necessary for the system.
Effectiveness

- We measure \textit{Flip Rate} of POE over TAPAS
  - This measures percentage of benchmarks that POE gets right but TAPAS gets wrong.

\[
FLIP\left(\frac{A}{B}\right) = \frac{|SUC\{A\} \cap FAIL\{B\}|}{|FAIL\{B\}|}
\]

POE can "fix" 39% of the benchmarks that TAPAS fails.

POE is effective in fixing wrong predictions of weakly supervised models.
Explainability: A User Study

- Evaluation -

We carry out a small user study on a comparison of the usability and explainability between TAPAS and POE.

- Task1 (Usability): Ask a question regarding the given visualization and evaluate which tool returns the accurate desired answers.

- Task2 (Explainability): Inspect the returned answer together with the explanation generated by POE, and tell whether the answer is well explained and aligns with the user intent.

As a result, the participants indicate in our results that POE is demonstrating better usability and explainability than TAPAS.
Discussions & Session Conclusions

• Discussions
  • Incomprehensive Questions
    • "What is highest change in income?" – typo
    • "In which year investors of all age groups took bigger risks?" – "bigger" should be "biggest"
  • Limitation of NLP Modules
    • "How many countries in Asia will have their economy improved based on majority votes?" – requires a knowledge base backend for inferring the implication of "countries in Asia"
    • "How many teams are in the Central Division?" – requires alignment with entities from the visualization to the range of "Central Division"

• Conclusions
  
  Poe is open-sourced and publicly available.
  
  https://github.com/chyanju/Poe
Conclusions and Proposals

Deduction-Powered Neural Program Synthesis: A Multi-Modal Perspective

- Lessons Learned
  - Deduction-Powered Program Synthesis
  - Neural Program Synthesis
  - Multi-Modal Program Synthesis
- Proposals
- Lessons Learned -

Deduction-Powered Program Synthesis

- **MARS**\(^{[1]}\)/**TRINITY**\(^{[2]}\)/**CONCORD**\(^{[3]}\)
  - Light-Weight SMT-Based Deduction + Partial Evaluation
  - Conflict-Driven Learning
- **HECATE**\(^{[4]}\)
  - Domain-Specific Symbolic Compilation
- **NGST2**\(^{[5]}\)
  - Trace Compatibility Checking with Concolic Execution and Bidirectional Reasoning
- **POE**\(^{[6]}\)
  - Abstract Program Synthesis
  - Optimal Program Synthesis

---


Neural Program Synthesis

- Lessons Learned -

MARS\(^1\)
\begin{itemize}
  \item Hybrid Neural Sequence Modeling of Programs
\end{itemize}

CONCORD\(^2\)
\begin{itemize}
  \item Deduction-Guided Reinforcement Learning
\end{itemize}

NGST2\(^3\)
\begin{itemize}
  \item Cognate Grammar Network (CGN)
\end{itemize}

\[\text{References:}\]
Multi-Modal Program Synthesis

- Lessons Learned -

- MARS\textsuperscript{[1]}
  - IO Example, Natural Language Description
  - Multi-Layer Specification

- POE\textsuperscript{[2]}
  - Neural Outputs, Natural Language Query, Visualization
  - Triangle Alignment Constraints

\textbf{and}

- CONCORD\textsuperscript{[3]}
  - IO Example, Programs
  - Importance Weighting

Proposals

• Deduction-Powered Neural Program Synthesis: A Multi-Modal Perspective

• Boosting Program Synthesis by Incorporation of:
  • Deduction-Powered Program Synthesis
    • More Customized and Precise: HECATE\textsuperscript{[1]}, NGST\textsuperscript{[2]}
    • E.g., Incrementality, Modularity, etc.
  • Neural Program Synthesis
    • More Semantic- and Syntactic- Aware: CONCORD\textsuperscript{[3]}, NGST\textsuperscript{[2]}
    • E.g., Fault Localization Networks, Meta Program Synthesis, etc.
  • Multi-Modal Program Synthesis
    • More Robust and Reliable: MARS\textsuperscript{[4]}, POE\textsuperscript{[5]}, CONCORD
    • E.g., User Interactions/Mistakes, Partial Annotations/Sketches, etc.

\textsuperscript{[1]} Tree Traversal Synthesis Using Domain-Specific Symbolic Compilation. Chen Y. et al. ASPLOS 2022.
\textsuperscript{[5]} Visualization Question Answering Using Introspective Program Synthesis. Chen Y. et al. PLDI 2022.
References and Related Works


