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

[1] **Tree Traversal Synthesis Using Domain-Specific Symbolic Compilation**. Yanju Chen, Junrui Liu, Yu Feng, Rastislav Bodik. *ASPLOS 2022.*

[2] **Visualization Question Answering Using Introspective Program Synthesis**. Yanju Chen, Xifeng Yan, Yu Feng. *PLDI 2022.*

01 Complex Software Systems

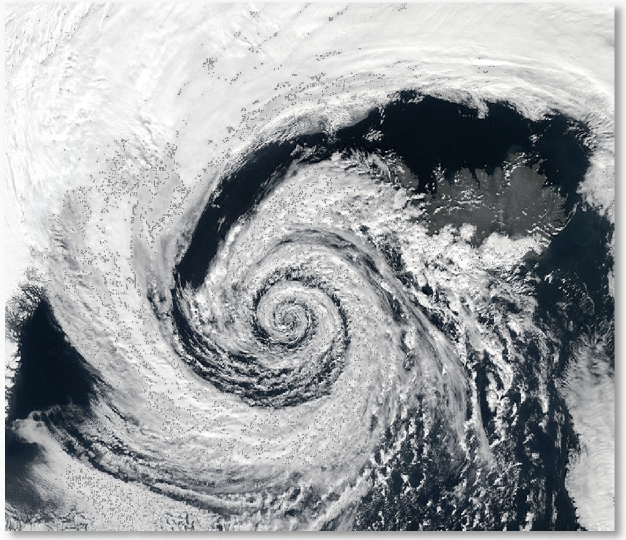
Complex Software Systems Around Us

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

Complex Systems

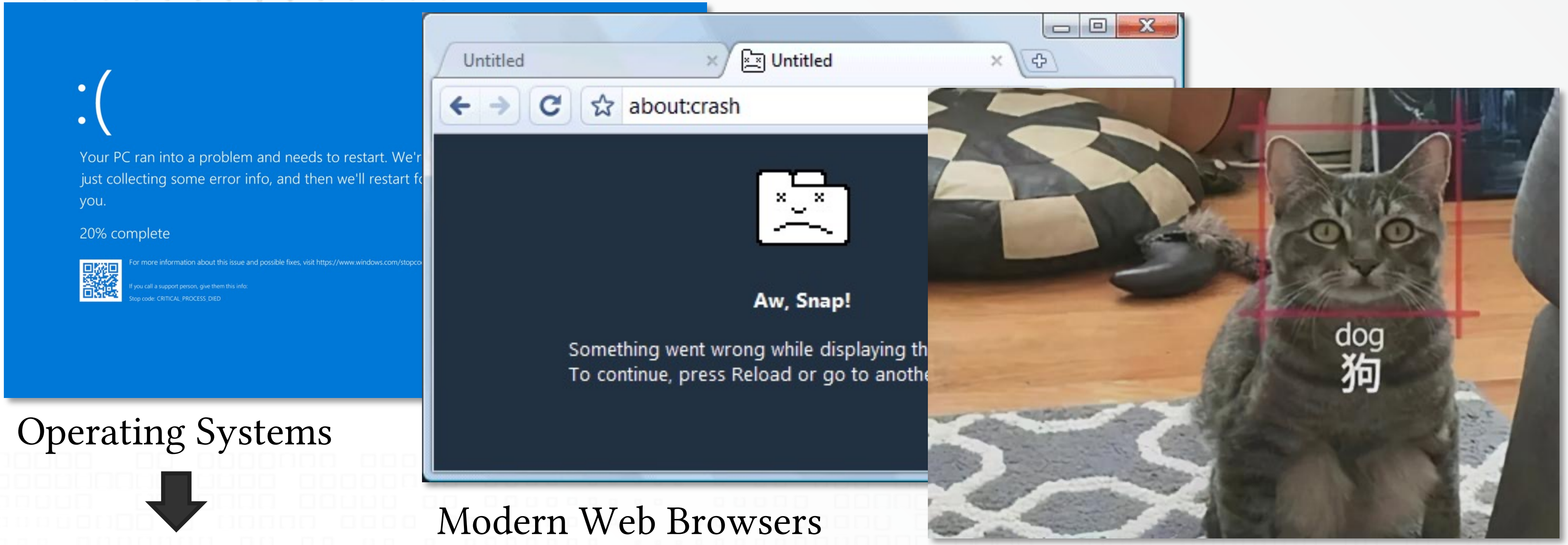
organisms

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]

[1] **Scaling symbolic evaluation for automated verification of systems code with Serval**. Nelson, L. et al. *SOSP 2019.*
[2] **Automated Reasoning for Web Page Layout**. Panchekha, P. et al. *OOPSLA 2016.*
[3] **"Why Should I Trust You?": Explaining the Predictions of Any Classifier**. Ribeiro, M.T. et al. *KDD 2016.*

02

Program Synthesis for

Modern Web Browsers

Tree Traversal Synthesis Using Domain-Specific Symbolic Compilation[1]

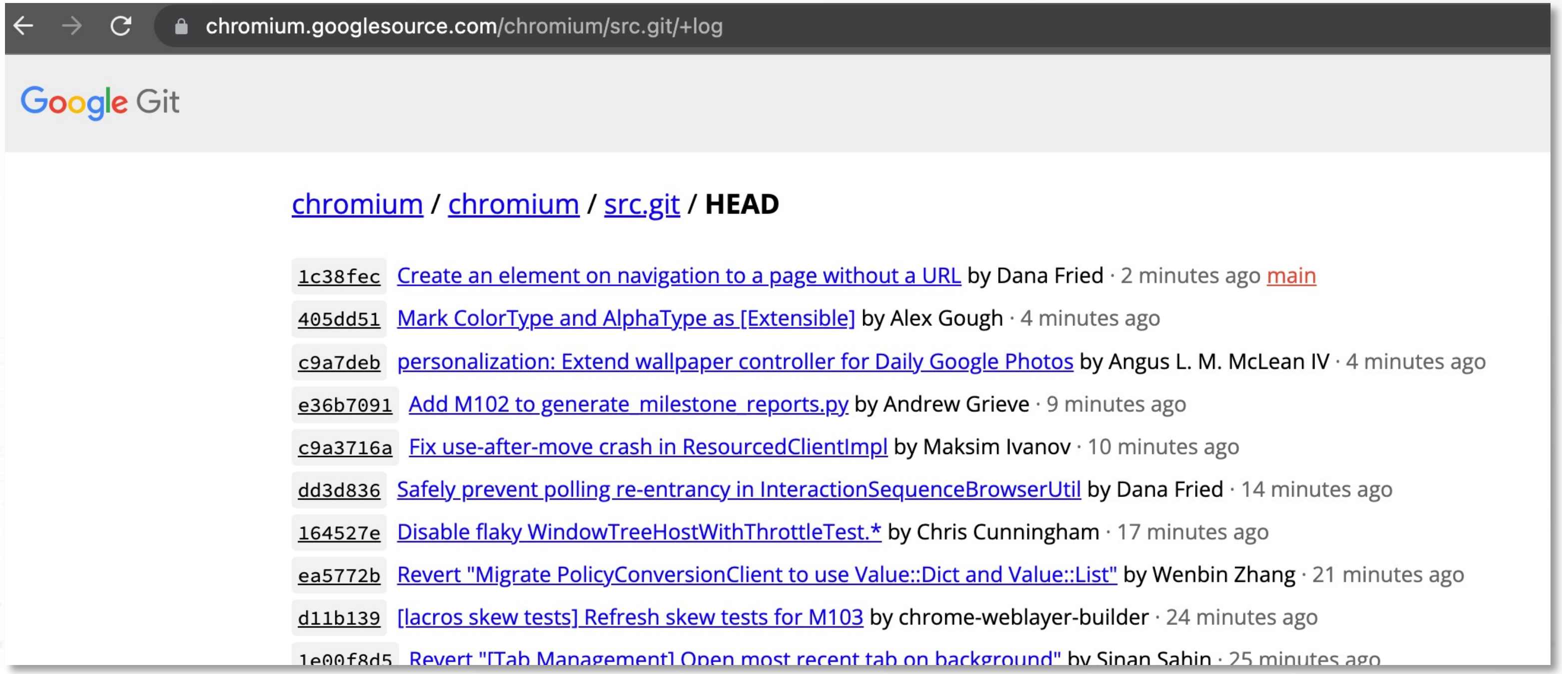
- Motivations
  - Modern Web Browsers
  - Tree Traversals
  - A Motivating Example
- Existing Approaches & Challenges
- Overview: HECATE
- Attribute Grammar & Traversal Language
- General-Purpose Symbolic Compilation
- Domain-Specific Symbolic Compilation
- Complexity Analysis
- Evaluation
  - GRAFTER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

[1] **Tree Traversal Synthesis Using Domain-Specific Symbolic Compilation**. Yanju Chen, Junrui Liu, Yu Feng, Rastislav Bodik. *ASPLOS 2022.*

- Motivations -

Modern Web Browsers

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



- Motivations -

Tree Traversals

Compilers

Web Browsers

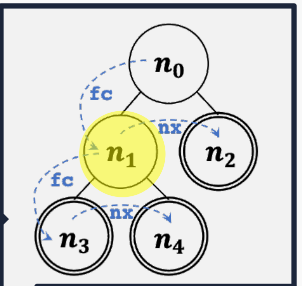
Numerical Computations

Tree traversals are widely used and play important roles.

- Motivations -

A Motivating Example

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



example tree

```
interface Box{
  input w0,h0: int;
  output w1,w,h1,h: int;
}
class Inner: Box{
  children {
    nx : Optional[Box];
    fc : Optional[Box];
  }
  rules {
    self.w := max( self.w0, fc.w1 );
    self.w1 := max( self.w, nx.w1 );
    self.h := max( self.h0, fc.h1 );
    self.h1 := self.h + nx.h1;
  }
}
class Leaf: Box{
  children {
    nx : Optional[Box];
  }
  rules {
    self.w := self.w0;
    self.w1 := max( self.w, nx.w1 );
    self.h := self.h0;
    self.h1 := self.h + nx.h1;
  }
}
```

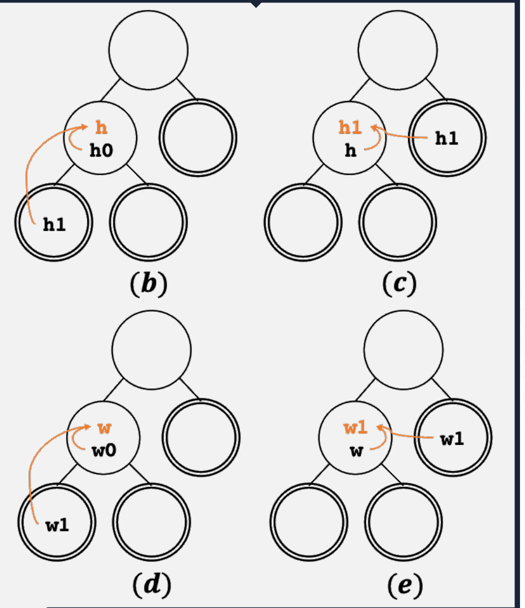
class definitions

```
traversal layout {
  case Inner{
    recur fc;
    recur nx;
    ℓ0;
    ℓ1;
    ℓ2;
    ℓ3;
  }
  case Leaf{
    recur nx;
    ℓ4;
    ℓ5;
    ℓ6;
    ℓ7
  }
}
```

symbolic traversal

```
traversal layout {
  case Inner{
    recur fc;
    recur nx;
    eval self.w;
    eval self.h;
    eval self.w1;
    eval self.h1;
  }
  case Leaf{
    recur nx;
    eval self.w;
    eval self.h;
    eval self.w1;
    eval self.h1;
  }
}
```

concrete traversal



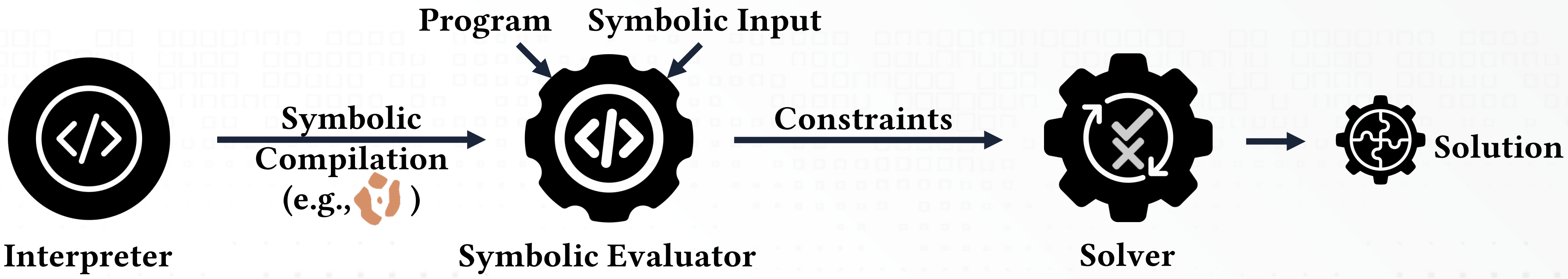
dependencies in n_1's attributes

Attribute Grammar

- Motivations -

Existing Approaches & Challenges

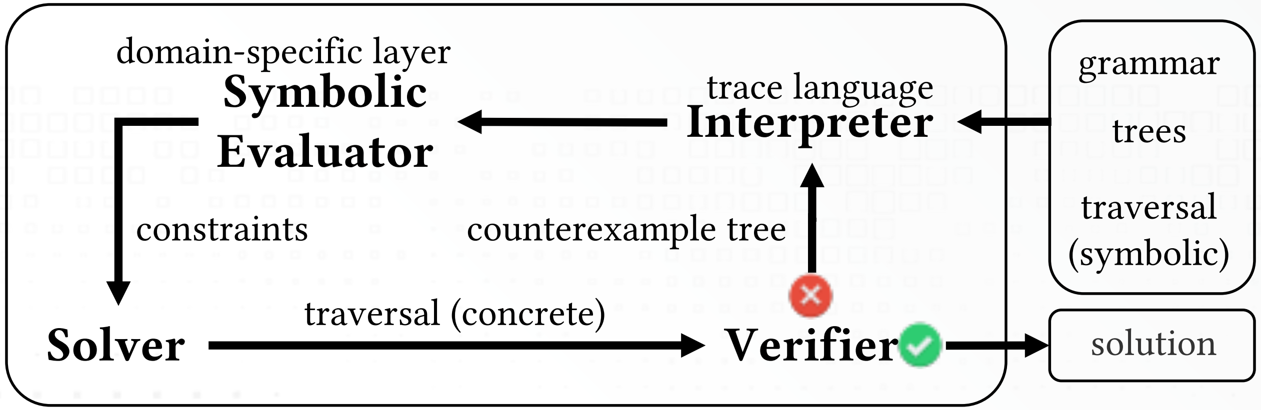
- Automata Based: TreeFuser[1] and GRAFTER[2]
  - Deterministic Rewrite Rules (Complex to Maintain)
- Synthesis Based: FTL[3]
  - Constraints Generated by Domain Experts (Manual and Error-Prone)
- General-Purpose Symbolic Compilation
  - Solver-Aided Programming Languages, e.g., Rosette[4]
  - Path Explosions & Complex Constraint System



[1] **TreeFuser: a framework for analyzing and fusing general recursive tree traversals**. Sakka, L. et al. *OOPSLA 2017*.
[2] **Sound, Fine-Grained Traversal Fusion for Heterogeneous Trees**. Sakka, L. et al. *PLDI 2019*.
[3] **Parallel Schedule Synthesis for Attribute Grammars**. Meyerovich, L. et al. *PPoPP 2013*.
[4] **A Lightweight Symbolic Virtual Machine for Solver-Aided Host Languages**. Torlak, E. et al. *PLDI 2014*.

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

$$
\begin{array}{lcl}
\langle \mathit{interface} \rangle & ::= & \text{interface } \langle \mathit{id} \rangle \ \{ \ (\langle \mathit{tup} \rangle;)^{*} \ \} \\
\langle \mathit{class} \rangle & ::= & \text{class } \langle \mathit{tup} \rangle \ \{ \ \langle \mathit{children} \rangle \ \langle \mathit{rules} \rangle \ \} \\
\langle \mathit{children} \rangle & ::= & \text{children} \ \{ \ (\langle \mathit{tup} \rangle;)^{*} \ \} \\
\langle \mathit{rules} \rangle & ::= & \text{rules} \ \{ \ (\langle \mathit{cstmt} \rangle;)^{*} \ \} \\
\langle \mathit{tup} \rangle & ::= & \langle \mathit{id} \rangle{:}\langle \mathit{id} \rangle(,\langle \mathit{id} \rangle)^{*} \\
\langle \mathit{sel} \rangle & ::= & \langle \mathit{id} \rangle(.\langle \mathit{id} \rangle)?.\langle \mathit{id} \rangle \\
\langle \mathit{expr} \rangle & ::= & \langle \mathit{const} \rangle \mid \langle \mathit{sel} \rangle \mid f(\ \langle \mathit{expr} \rangle^{*} \ ) \\
 & \mid & \langle \mathit{expr} \rangle \ \langle \mathit{op} \rangle \ \langle \mathit{expr} \rangle \mid \text{fold}(\ \langle \mathit{expr} \rangle + \ ) \\
 & \mid & \text{if } \langle \mathit{expr} \rangle \text{ then } \langle \mathit{expr} \rangle \text{ else } \langle \mathit{expr} \rangle \\
\langle \mathit{cstmt} \rangle & ::= & \langle \mathit{sel} \rangle := \langle \mathit{expr} \rangle \\
\langle \mathit{op} \rangle & ::= & + \mid - \mid \times \mid \div \mid \ldots
\end{array}
$$

$f \in$ **functions** $\quad \langle \mathit{const} \rangle \in$ **constants** $\quad \langle \mathit{id} \rangle \in$ **identifiers**

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

$$
\begin{array}{lcl}
\langle \mathit{traversal} \rangle & ::= & \text{traversal } \langle \mathit{id} \rangle \ \{ \ \langle \mathit{case} \rangle^{*} \ \} \\
\langle \mathit{case} \rangle & ::= & \text{case } \langle \mathit{id} \rangle \ \{ \ (\langle \mathit{tstmt} \rangle;)^{*} \ \} \\
\langle \mathit{recur} \rangle & ::= & \text{recur } \langle \mathit{node} \rangle \\
\langle \mathit{iterate} \rangle & ::= & \text{iterate} \ \{ \ (\langle \mathit{tstmt} \rangle;)^{*} \ \} \\
\langle \mathit{parallel} \rangle & ::= & \text{parallel} \ \{ \ (\langle \mathit{tstmt} \rangle;)^{*} \ \} \\
\langle \mathit{eval} \rangle & ::= & \text{eval } \langle \mathit{cstmt} \rangle \\
\langle \mathit{tstmt} \rangle & ::= & \iota \mid \langle \mathit{recur} \rangle \mid \langle \mathit{iterate} \rangle \mid \langle \mathit{eval} \rangle
\end{array}
$$

$\langle \mathit{id} \rangle \in$ **identifiers** $\quad \langle \mathit{node} \rangle \in$ **nodes**

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

$$
\begin{aligned}
&(\sigma(\texttt{none}, \iota_2) \implies \mathit{true}) \\
\lor\ &(\sigma(\texttt{Inner.w1}, \iota_2) \implies \delta(\zeta(n_1, \texttt{self.w}), t) \land \delta(\zeta(n_1, \texttt{nx.w1}), t) \\
&\quad \land \neg\delta(\zeta(n_1, \texttt{self.w1}), t)) \\
\lor\ &(\sigma(\texttt{Inner.w}, \iota_2) \implies \delta(\zeta(n_1, \texttt{self.w0}), t) \land \delta(\zeta(n_1, \texttt{fc.w1}), t) \\
&\quad \land \neg\delta(\zeta(n_1, \texttt{self.w}), t)) \\
\lor\ &(\sigma(\texttt{Inner.h1}, \iota_2) \implies \delta(\zeta(n_1, \texttt{self.h}), t) \land \delta(\zeta(n_1, \texttt{nx.h1}), t) \\
&\quad \land \neg\delta(\zeta(n_1, \texttt{self.h1}), t)) \\
\lor\ &(\sigma(\texttt{Inner.h}, \iota_2) \implies \delta(\zeta(n_1, \texttt{self.h0}), t) \land \delta(\zeta(n_1, \texttt{fc.h1}), t) \\
&\quad \land \neg\delta(\zeta(n_1, \texttt{self.h}), t))
\end{aligned}
$$

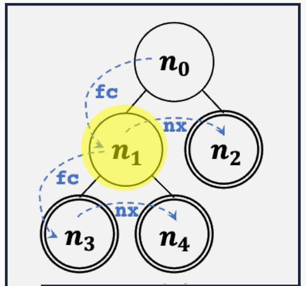
"choose one to schedule"

"all dependencies should have been ready"

"target attribute has not been scheduled"

```
traversal layout {
  case Inner{
    recur fc;
    recur nx;
    ι0;
    ι1;
    ι2;
    ι3;
  }
  case Leaf{
    recur nx;
    ι4;
    ι5;
    ι6;
    ι7
  }
}
```

symbolic traversal



example tree

```
class Inner: Box{
  children {
    nx : Optional[Box];
    fc : Optional[Box];
  }
  rules {
    self.w := max( self.w0, fc.w1 );
    self.w1 := max( self.w, nx.w1 );
    self.h := max( self.h0, fc.h1 );
    self.h1 := self.h + nx.h1;
  }
}
```

class definitions

Number of timesteps grows as example trees become larger, which increases the complexity.

  - Auxiliary Constraints

$$\forall \iota. (\bigvee_{a_0} \bigwedge_{a \neq a_0} \neg\sigma(a, \iota) \land \sigma(a_0, \iota)) \lor (\bigwedge_{a} \neg\sigma(a, \iota)).$$

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

$$\forall a. \bigvee_{\iota_0} \bigwedge_{\iota \neq \iota_0} \neg\sigma(a, \iota) \land \sigma(a, \iota_0).$$

- Every rule should be used by only one slot.

- Synthesis Using HECATE -

Domain-Specific Symbolic Compilation

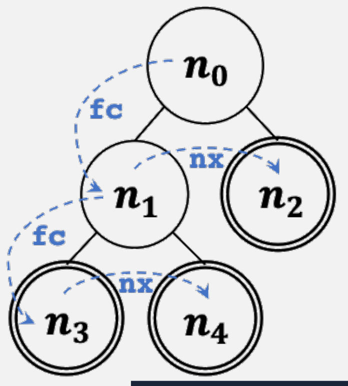
- **[Traversal]** Given a tree, a traversal defines a total order relation $\prec$ over the set of all locations of the tree.

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

$$n_4.\mathtt{w} \prec n_4.\mathtt{h} \prec n_4.\mathtt{w1} \prec n_4.\mathtt{h1} \prec n_3.\mathtt{w} \prec n_3.\mathtt{h} \prec n_3.\mathtt{w1} \prec n_3.\mathtt{h1}$$
$$\prec n_1.\mathtt{w} \prec n_1.\mathtt{h} \prec n_1.\mathtt{w1} \prec n_1.\mathtt{h1} \prec n_2.\mathtt{w} \prec n_2.\mathtt{h} \prec n_2.\mathtt{w1} \prec n_2.\mathtt{h1}$$
$$\prec n_0.\mathtt{w} \prec n_0.\mathtt{h} \prec n_0.\mathtt{w1} \prec n_0.\mathtt{h1}$$

```
traversal layout {
  case Inner{
    recur fc;
    recur nx;
    ι0;
    ι1;
    ι2;
    ι3;
  }
  case Leaf{
    recur nx;
    ι4;
    ι5;
    ι6;
    ι7
  }
}
```

symbolic traversal



example tree

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

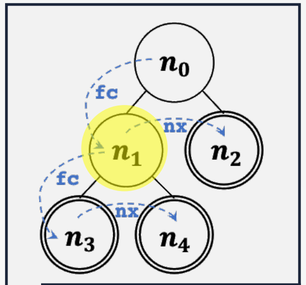
- A Symbolic Trace Language

Operation	Description
`(choose` $[a_1, \dots a_n]$`)`	choose one from the attributes
`(alloc)`	returns a fresh concrete location
`(read` $n.a$`)`	logs a read from $n.a$
`(write` $n.a$`)`	logs a write to $n.a$

(assume σ(Inner.h, ι_2)
(read n_1.h0) (read n_3.h1) (write n_1.h))

```
traversal layout {
  case Inner{
    recur fc;
    recur nx;
    ι0;
    ι1;
    ι2;
    ι3;
  }
  case Leaf{
    recur nx;
    ι4;
    ι5;
    ι6;
    ι7
  }
}
```

symbolic traversal



example tree

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

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

where all entries are integers and in particular $x_j \in \{0,1\}$.

- Synthesis Using HECATE -

Domain-Specific Symbolic Compilation

(assume σ(Inner.h, ι_2) (read n_1.h0) (read n_3.h1) (write n_1.h))

- Constraint System
  - Dependency Constraints

$$\sigma(\texttt{Inner.h}, \iota_2) \leq \sum_{t_0<t} \kappa[n_1.\texttt{h0}, t_0] = \sigma(\texttt{Inner.h0}, \iota_0) + \sigma(\texttt{Inner.h0}, \iota_1), \quad \text{(read for } n_1.\texttt{h0})$$

$$\sigma(\texttt{Inner.h}, \iota_2) \leq \sum_{t_0<t} \kappa[n_3.\texttt{h1}, t_0] = \sigma(\texttt{Leaf.h1}, \iota_4) + \sigma(\texttt{Leaf.h1}, \iota_5) + \sigma(\texttt{Leaf.h1}, \iota_6) + \sigma(\texttt{Leaf.h1}, \iota_7), \quad \text{(read for } n_3.\texttt{h1})$$

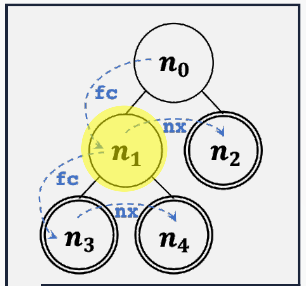
"n_1.h0 should have been scheduled somewhere before the current corresponding location"

  - Validity Constraints

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

$$\forall \iota. \sum_a \sigma(a, \iota) \leq 1,$$ - Every slot should be filled with at most one rule.

$$\forall a. \sum_\iota \sigma(a, \iota) = 1.$$ - Every rule should be used by only one slot.



example tree

```
class Inner: Box{
  children {
    nx : Optional[Box];
    fc : Optional[Box];
  }
  rules {
    self.w := max( self.w0, fc.w1 );
    self.w1 := max( self.w, nx.w1 );
    self.h := max( self.h0, fc.h1 );
    self.h1 := self.h + nx.h1;
  }
}
```

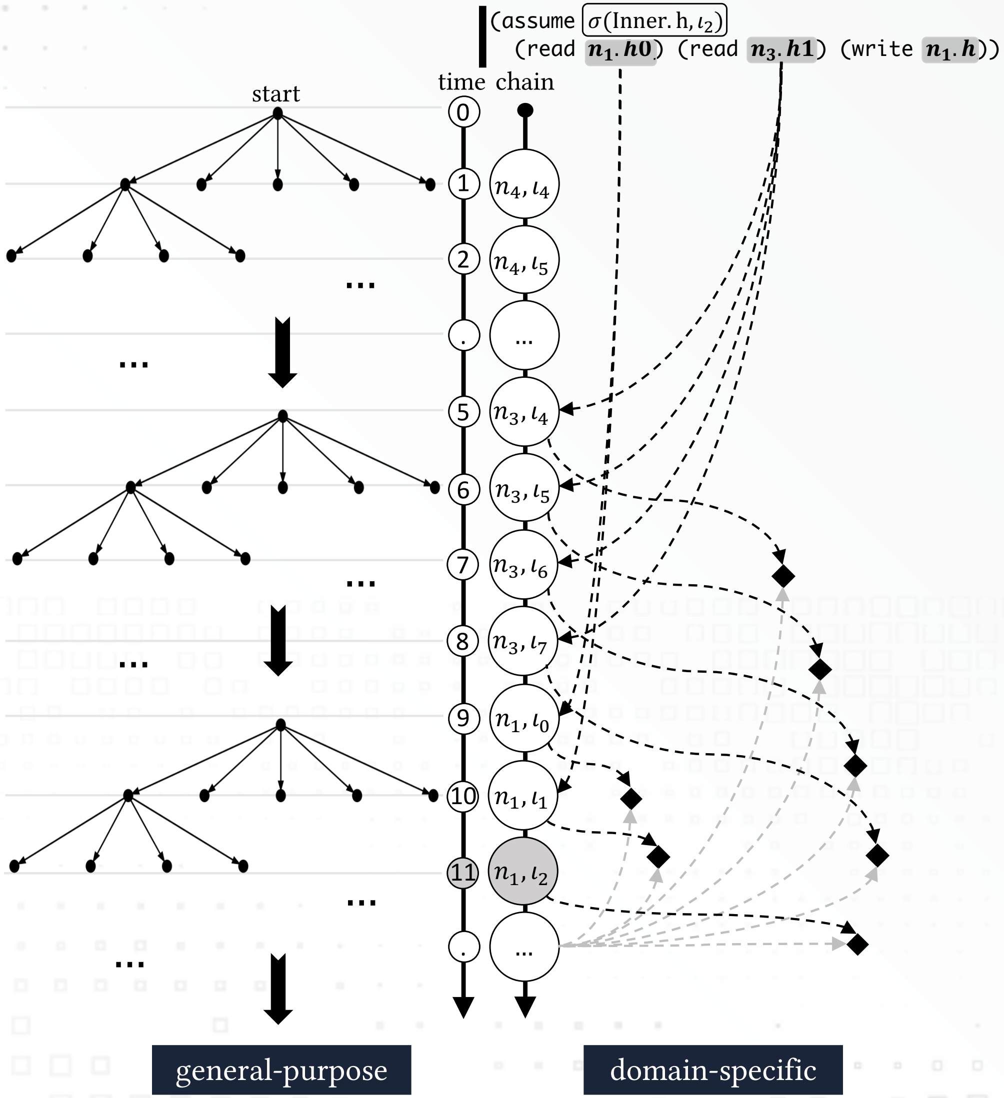
class definitions

```
traversal layout {
  case Inner{
    recur fc;
    recur nx;
    ι0;
    ι1;
    ι2;
    ι3;
  }
  case Leaf{
    recur nx;
    ι4;
    ι5;
    ι6;
    ι7
  }
}
```

symbolic traversal

- Synthesis Using HECATE -

Complexity Analysis



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?

- Evaluation -

Comparison against Grafter[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$^{\mathbb{G}}$ (with general-purpose encoding). The table shows total synthesis time (synthesis + verification) in second.

Benchmark	# of Rules	GRAFTER	HECATE	HECATE$^{\mathbb{G}}$
BinaryTree	16	2.6	1.1	3.2
FMM	14	7.6	1.0	1.6
Piecewise	12	12.6	2.1	3.1
AST	136	151.7	20.6	73.4
RenderTree	50	62.0	4.1	10.1

[1] **Sound, Fine-Grained Traversal Fusion for Heterogeneous Trees**. Sakka, L. et al. *PLDI 2019.*

- Evaluation -

A Case Study: RenderTree

- 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 $^{\mathbb{L}}$: Sequential, Linked List
  - HECATE $^{\mathbb{V}}$: Sequential, Vector
  - HECATE $^{\mathbb{P}}$: Parallel, Vector

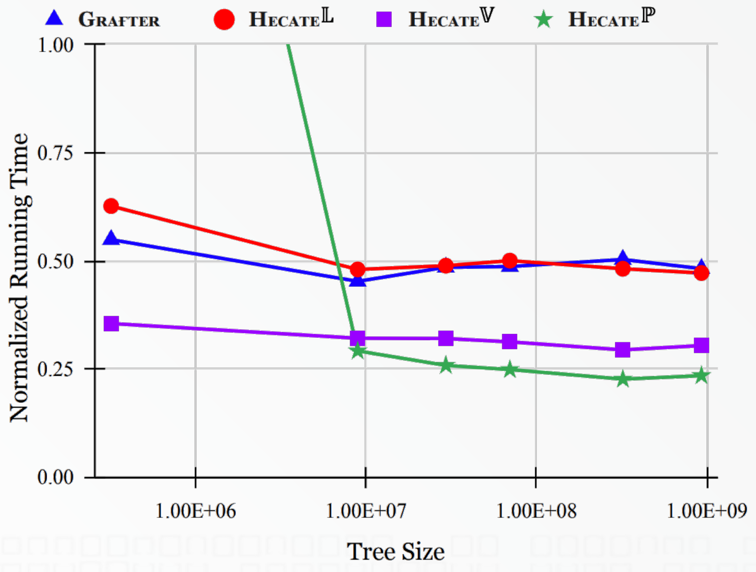


Figure 11: Running time of fused traversals compared to the unfused baseline.

With minimal efforts, Hecate can effectively explore traversals of different design choices.

- Evaluation -

Synthesizing Layout Engine in FTL[1]

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

Name	# of Rules
CSS-float	192
CSS-margin	178
CSS-full	244

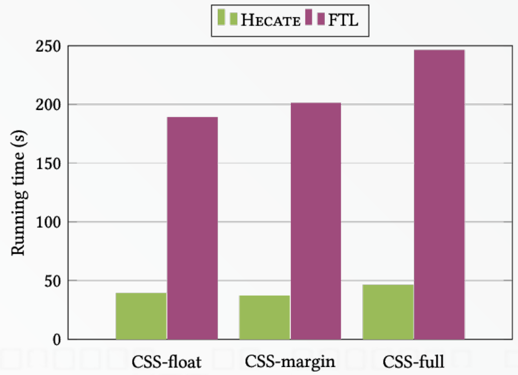


Figure 15: Comparison against FTL: benchmark statistics (left) and results (right).

[1] **Parallel Schedule Synthesis for Attribute Grammars**. Meyerovich, L. et al. *PPoPP 2013*.

Session Conclusions

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

https://github.com/chyanju/Hecate

Hecate is open-sourced and publicly available.

03

Program Synthesis for

Deep Learning Systems

Visualization Question Answering Using Introspective Program Synthesis[1]

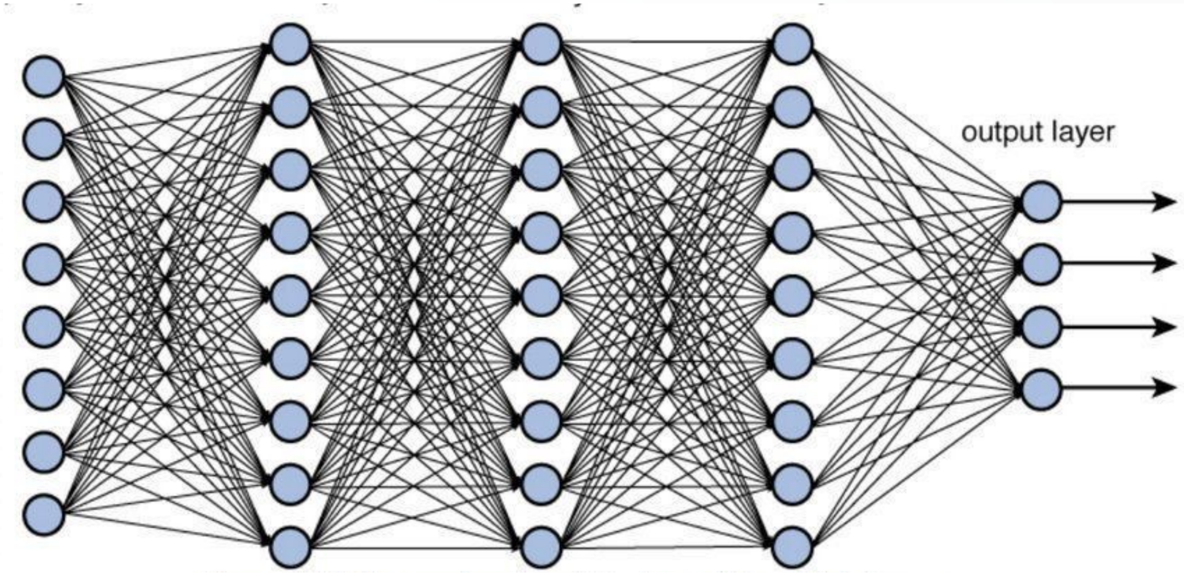
- Deep Learning Systems
- VQA: A Motivating Example
- Existing Approaches & Challenges
- Observations
- Overview: POE
- A Walkthrough of POE
- System Workflow in POE
- Introspective Program Synthesis
- Abstract Program Synthesis with Noisy Specification
- Optimal Program Synthesis for Explanation Refinement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

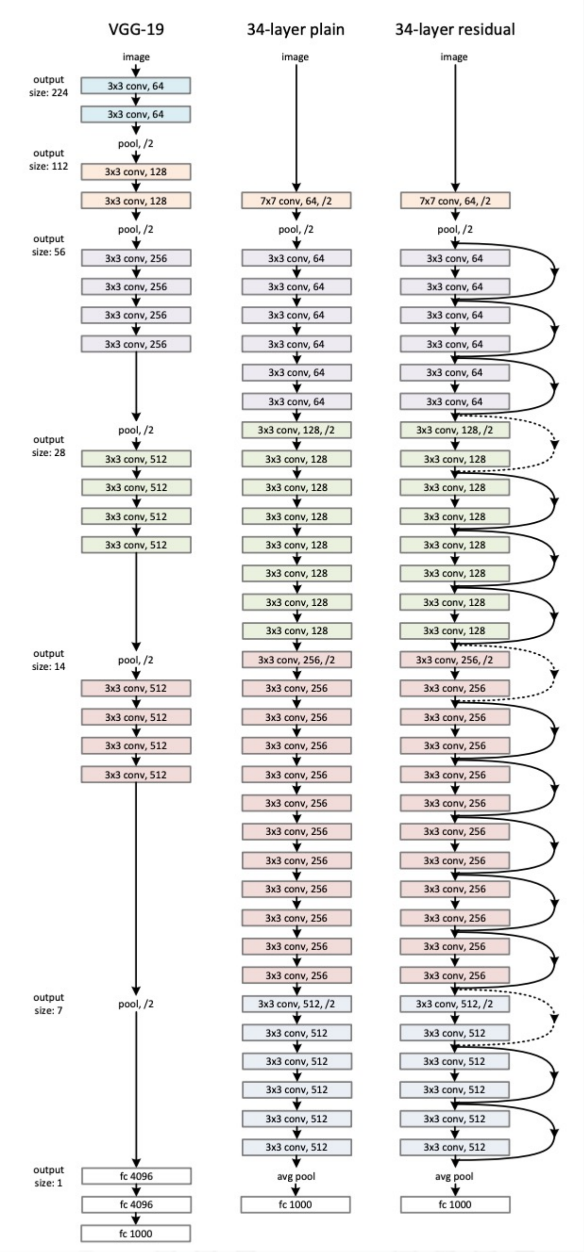
[1] **Visualization Question Answering Using Introspective Program Synthesis**. Yanju Chen, Xifeng Yan, Yu Feng. *PLDI 2022.*

- Motivations -

Deep Learning Systems

- Data-Driven
- "Black Box"
- Deep & Large-Scale
- ...





- Motivations -

VQA: A Motivating Example

(Visualization Question Answering)

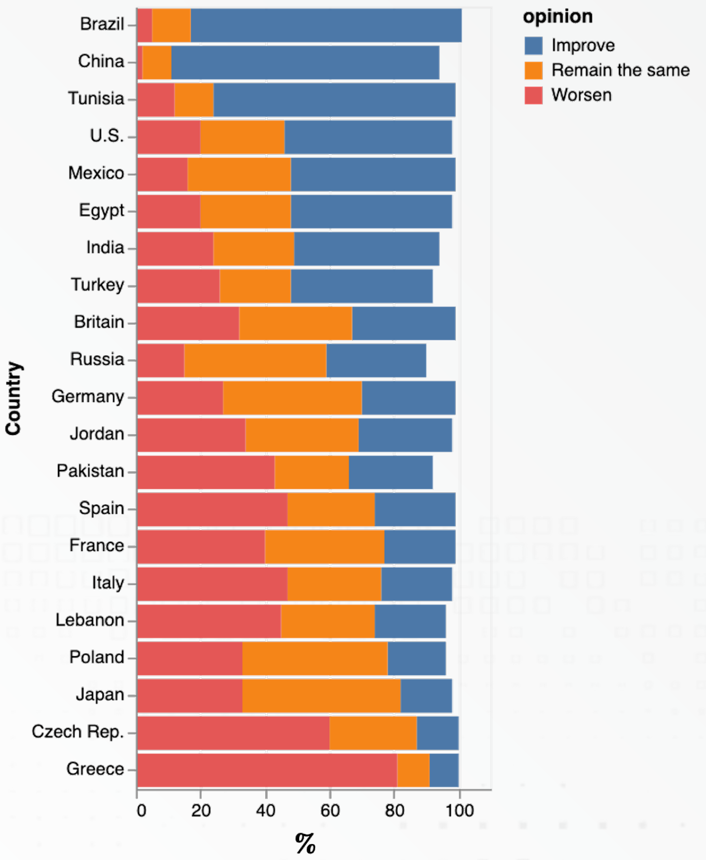
- Given a stacked bar chart that represents opinions for future economic growth for different countries, a user describes her query based on the visualization in natural language:

Query

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.

Visualization



- Motivations -

Existing Approaches & Challenges

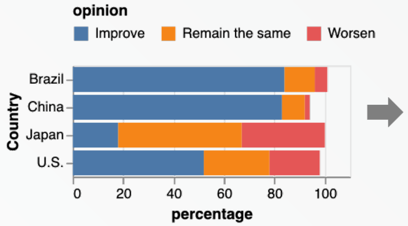
- Pipelined System: VisQA[1]
  - The first one that specializes for VQA tasks
  - Errors propagate between different system modules
- Fully Supervised Machine Learning: SmBoP[2], NL2code[3]
  - Requires manual annotated logic forms / programs as supervised training data
  - Targeted at Table Question 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

[1] **Answering Questions about Charts and Generating Visual Explanations**. Kim, D.H. et al. *CHI 2020.*
[2] **SmBoP: Semi-autoregressive Bottom-up Semantic Parsing**. Rubin, O. et al. *NAACL 2021.*
[3] **A Syntactic Neural Model for General-Purpose Code Generation**. Yin, P. et al. *ACL 2017.*
[4] **TaPas: Weakly Supervised Table Parsing via Pre-training**. Herzig, J. et al. *ACL 2020.*

- Motivations -

Observations

- 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.



Country	Improve	Worsen	Remain the same
Brazil	84	5	12
China	83	2	9
Japan	18	33	49
U.S.	52	20	26

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

- 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.

Query

Which country has highest Improve value?

"Brazil", "Japan", "China", "U.S.", ...

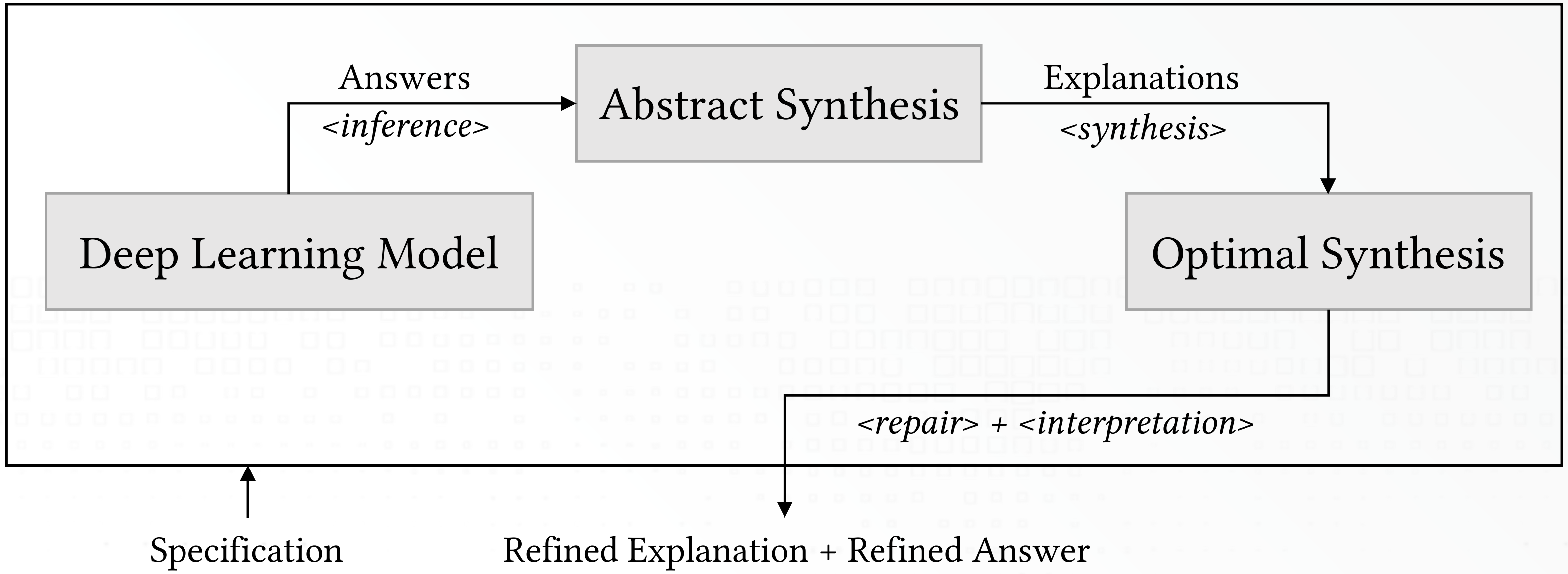
```
project(aggregate(I, null, max, ◇), ["Country"])

project(aggregate(I, null, ◇, ◇), ["Country"])
```

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

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.

- Synthesis Using POE -

A Walkthrough of POE

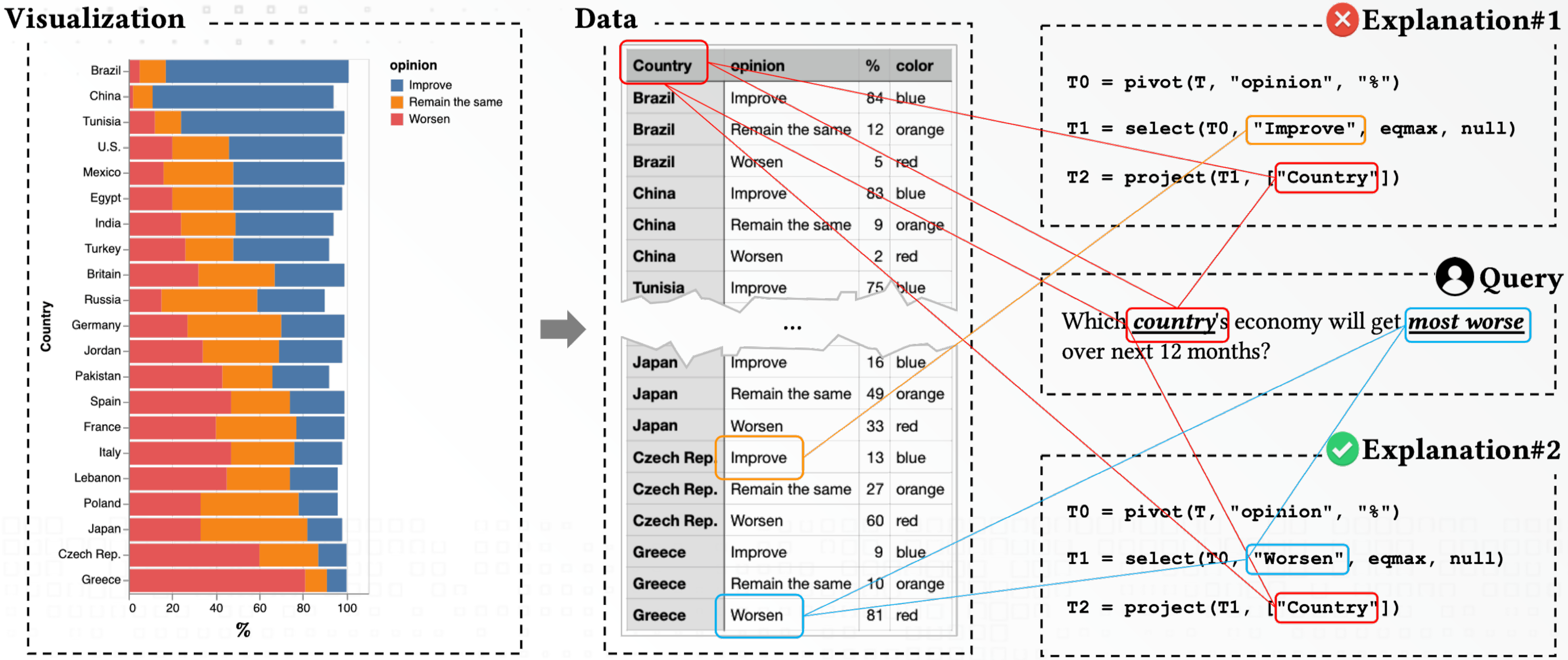


Figure 2. A motivating example on data of opinions for future economic growth for different countries. Left: A visualization of stacked bar chart for illustrating the data distribution; Middle: The corresponding table format of the data; Right: Example checking semantic consistency between three parties: 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.

- Synthesis Using PoE -

A Walkthrough of PoE

- Original TaPas Outputs:

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

- PoE's Abstract Program Synthesis Outputs:

```
project(select(pivot(T, ◇, ◇), ◇, ◇, ◇), ◇)
project(select(T, ◇, ◇, ◇), ◇)
...
```

- PoE's Optimal Program Synthesis Outputs:

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

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

Query

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

Data

Country	opinion	%	color
Brazil	Improve	84	blue
Brazil	Remain the same	12	orange
Brazil	Worsen	5	red
China	Improve	83	blue
China	Remain the same	9	orange
China	Worsen	2	red
Tunisia	Improve	75	blue
...			
Japan	Improve	16	blue
Japan	Remain the same	49	orange
Japan	Worsen	33	red
Czech Rep.	Improve	13	blue
Czech Rep.	Remain the same	27	orange
Czech Rep.	Worsen	60	red
Greece	Improve	9	blue
Greece	Remain the same	10	orange
Greece	Worsen	81	red

```
⟨Table⟩   ::= project( ⟨Table⟩, ⟨ColList⟩ )
            |  select( ⟨Table⟩, ⟨BoolOp⟩, ⟨ColInt⟩, ⟨ConstVal⟩ )
            |  pivot( ⟨Table⟩, ⟨ColInt⟩, ⟨ColInt⟩ )
            |  aggregate( ⟨Table⟩, ⟨ColList⟩, ⟨AggrOp⟩, ⟨ColInt⟩ )
⟨AggrOp⟩  ::= count | min | max | sum | mean
⟨BoolOp⟩  ::= < | <= | == | >= | > | != | eqmax | eqmin
```

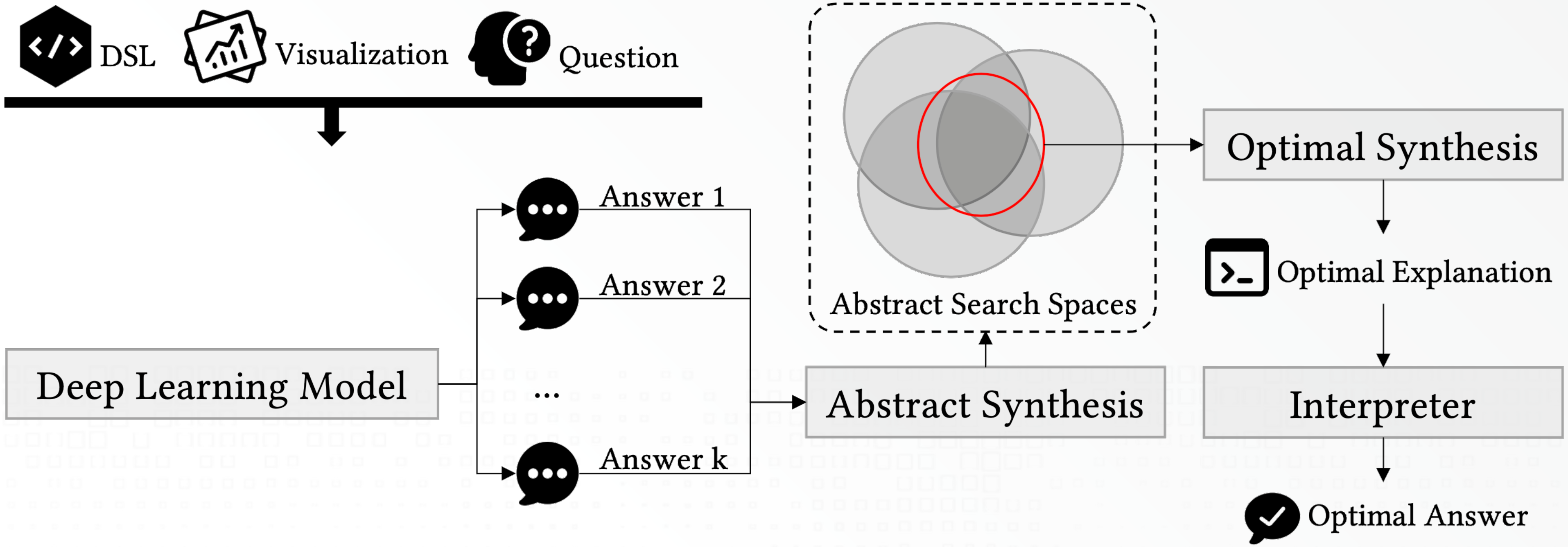
$\langle Table\rangle \in$ **tables**, $\langle ConstVal\rangle \in$ **constants**

$\langle ColInt\rangle \in$ **columns**, $\langle ColList\rangle \in$ **columns**^{n}

Figure 3. Syntax of a toy DSL for data wrangling.

- Synthesis Using POE -

System Workflow in POE



- Synthesis Using PoE -

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 π 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 $S \overset{*}{\Rightarrow} P$ and P optimizes the following objectives $\mathcal{O}$:

$$\begin{aligned} P^* &= \arg\max_P J_{\mathcal{T},\pi}(P) \\ &= \arg\max_P \sum_{o \in \mathcal{O}} \theta_o \cdot o(I, Q, \mathcal{A}, P), \end{aligned}$$

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

- Synthesis Using POE -

Abstract Program Synthesis with Noisy Specification

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

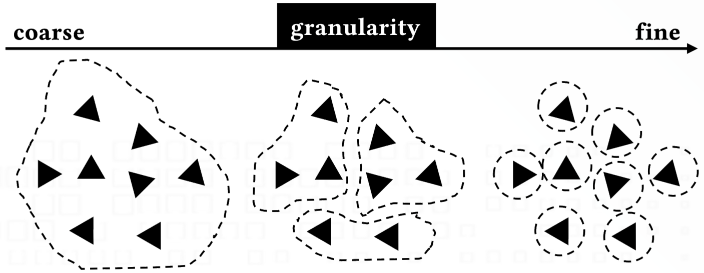
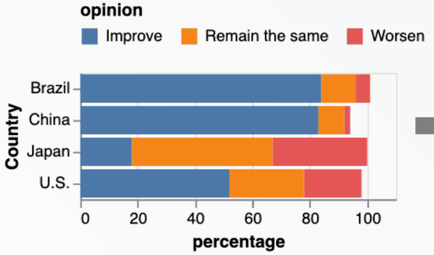


Figure 6. Different granularities that affect the algorithm search space. An input-output pair is denoted by a triangle.



Country	Improve	Worsen	Remain the same
Brazil	84	5	12
China	83	2	9
Japan	18	33	49
U.S.	52	20	26

Model Outputs: “Brazil”, “Japan”, “China”, “U.S.”

Abstract Synthesis Breakdown:

$\diamond$
⇨ feasible for all examples

project($\diamond$)
⇨ feasible for all examples

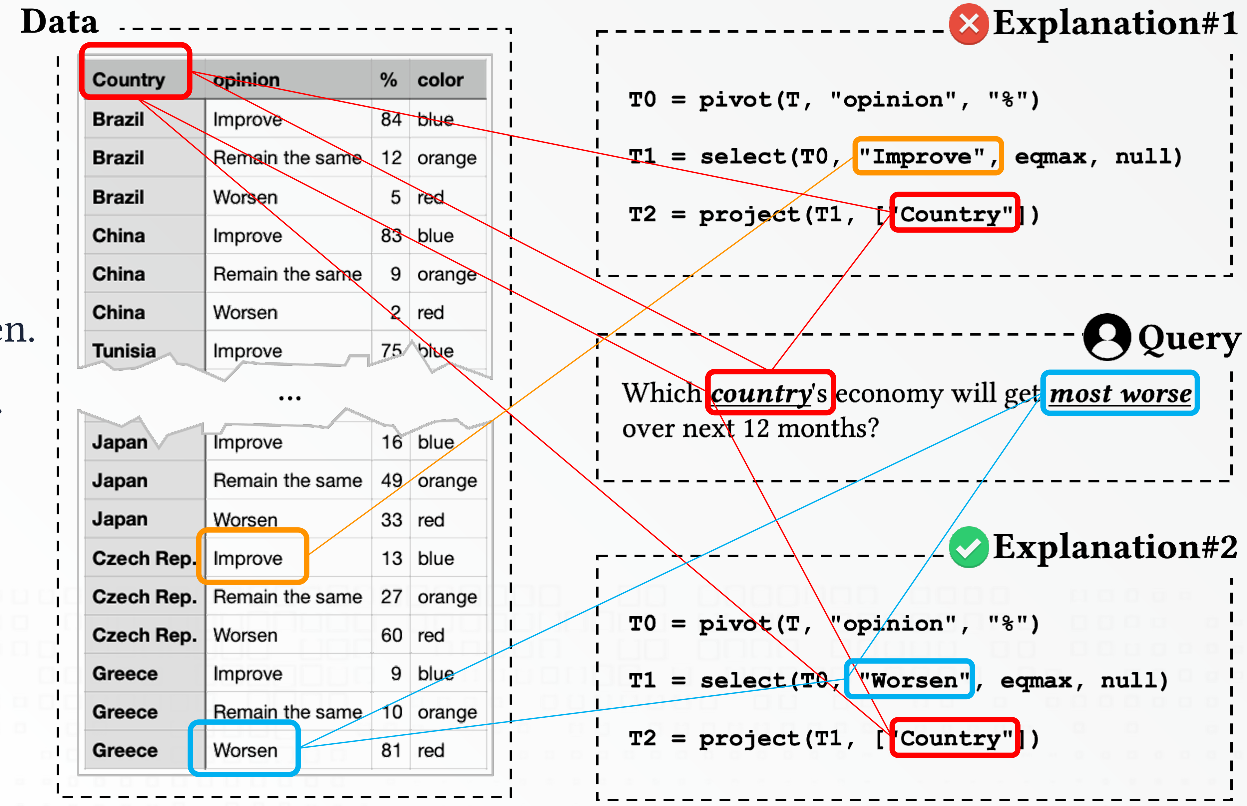
project(aggregate(I, null, $\diamond_0$, $\diamond_1$), ["Country"])
⇨ feasible for "Brazil", "Japan", "China"

project(aggregate(I, null, max, $\diamond_1$), ["Country"])
⇨ feasible for "Brazil", "Japan"

- Synthesis Using POE -

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.



$\mathtt{project}(\mathtt{select}(\mathtt{pivot}(\mathrm{T}, \diamond_0, \diamond_1), \diamond_2, \diamond_3, \diamond_4), \diamond_5)$

- Synthesis Using POE -

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)

$$\mathrm{NSYN}(\text{“high”}, \text{“highest”}) > \mathrm{NSYN}(\text{“high”}, \text{“low”})$$

- Soft Constraints / Objective Function

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

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

[1] **Answering Questions about Charts and Generating Visual Explanations**. Kim, D.H. et al. *CHI 2020.*

- Evaluation -

Performance

- Comparison against TAPAS[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

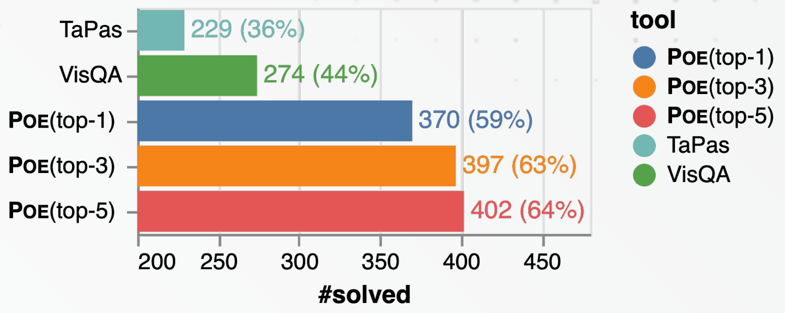


Figure 7. Performance comparison between the original pipeline from VisQA (baseline), TAPAS and POE.

POE can greatly boost performance of weakly supervised models.

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

question type	total	VisQA (baseline)	TAPAS	POE (top-1)
retrieval	183 (29%)	101 (55%)	98 (54%)	**123 (67%)**
comparison	87 (14%)	50 (57%)	0 (0%)	**71 (82%)**
aggregation	253 (40%)	92 (36%)	119 (47%)	**161 (64%)**
other	106 (17%)	**31 (29%)**	12 (11%)	15 (14%)
total	629 (100%)	274 (44%)	229 (36%)	**370 (59%)**

POE is effective across different types of benchmarks.

[1] **TaPas: Weakly Supervised Table Parsing via Pre-training**. Herzig, J. et al. *ACL 2020*.
[2] **Answering Questions about Charts and Generating Visual Explanations**. Kim, D.H. et al. *CHI 2020*.

- Evaluation -

Ablation Study

- Variants of POE
  - POE$^{\mathcal{O}}$: only performs optimal synthesis on the full search space
  - POE$^{\mathcal{A}}$: 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.

variant	TAPAS	POE	POE$^{\mathcal{A}}$	POE$^{\mathcal{O}}$
solved	229	370	194	357
delta (%)	(+0%)	(+23%)	(-5%)	(+21%)
#timeout	-	36	586	58

Both procedures are necessary for the system.

- Evaluation -

Effectiveness

- We measure ***Flip Rate*** of POE over TAPAS
  - This measures percentage of benchmarks that POE gets right but TAPAS gets wrong.

$$FLIP(\frac{A}{B}) = \frac{|SUCC(A) \cap FAIL(B)|}{|FAIL(B)|}$$

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

POE is effective in fixing wrong predictions of weakly supervised models.

- Evaluation -

Explainability: A User Study

- 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 highestt 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

https://github.com/chyanju/Poe

Poe is open-sourced and publicly available.

04 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

[1] **Maximal Multi-Layer Specification Synthesis**. Chen Y. et al. *FSE 2019*.
[2] **Trinity: An Extensible Synthesis Framework for Data Science**. Martins R. et al. *VLDB 2019*.
[3] **Program Synthesis Using Deduction-Guided Reinforcement Learning**. Chen Y. et al. *CAV 2020*.
[4] **Tree Traversal Synthesis Using Domain-Specific Symbolic Compilation**. Chen Y. et al. *ASPLOS 2022*.
[5] **Automated Transpilation of Imperative to Functional Code Using Neural-Guided Program Synthesis**. Mariano B. et al. *OOPSLA 2022*.
[6] **Visualization Question Answering Using Introspective Program Synthesis**. Chen Y. et al. *PLDI 2022*.

- Lessons Learned -

Neural Program Synthesis

- MARS[1]
  - Hybrid Neural Sequence Modeling of Programs
- CONCORD[2]
  - Deduction-Guided Reinforcement Learning
- NGST2[3]
  - Cognate Grammar Network (CGN)

[1] **Maximal Multi-Layer Specification Synthesis**. Chen Y. et al. *FSE 2019*.
[2] **Program Synthesis Using Deduction-Guided Reinforcement Learning**. Chen Y. et al. *CAV 2020*.
[3] **Automated Transpilation of Imperative to Functional Code Using Neural-Guided Program Synthesis**. Mariano B. et al. *OOPSLA 2022*.

- Lessons Learned -

Multi-Modal Program Synthesis

- MARS[1]
  - IO Example, Natural Language Description
  - Multi-Layer Specification
- POE[2]
  - Neural Outputs, Natural Language Query, Visualization
  - Triangle Alignment Constraints

………… *and*

- *CONCORD*[3]
  - IO Example, Programs
  - Importance Weighting

[1] **Maximal Multi-Layer Specification Synthesis**. Chen Y. et al. *FSE 2019.*
[2] **Visualization Question Answering Using Introspective Program Synthesis**. Chen Y. et al. *PLDI 2022.*
[3] **Program Synthesis Using Deduction-Guided Reinforcement Learning**. Chen Y. et al. *CAV 2020.*

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[1], NGST2[2]
    - E.g., Incrementality, Modularity, etc.
  - Neural Program Synthesis
    - More Semantic- and Syntactic- Aware: CONCORD[3], NGST2
    - E.g., Fault Localization Networks, Meta Program Synthesis, etc.
  - Multi-Modal Program Synthesis
    - More Robust and Reliable: MARS[4], POE[5], CONCORD
    - E.g., User Interactions/Mistakes, Partial Annotations/Sketches, etc.

[1] **Tree Traversal Synthesis Using Domain-Specific Symbolic Compilation**. Chen Y. et al. *ASPLOS 2022.*
[2] **Automated Transpilation of Imperative to Functional Code Using Neural-Guided Program Synthesis**. Mariano B. et al. *OOPSLA 2022.*
[3] **Program Synthesis Using Deduction-Guided Reinforcement Learning**. Chen Y. et al. *CAV 2020.*
[4] **Maximal Multi-Layer Specification Synthesis**. Chen Y. et al. *FSE 2019.*
[5] **Visualization Question Answering Using Introspective Program Synthesis**. Chen Y. et al. *PLDI 2022.*

References and Related Works

[1] Scaling symbolic evaluation for automated verification of systems code with Serval. Luke Nelson, James Bornholt, Ronghui Gu, Andrew Baumann, Emina Torlak, and Xi Wang. SOSP 2019.

[2] Automated Reasoning for Web Page Layout. Pavel Panchekha, Emina Torlak. OOPSLA 2016.

[3] "Why Should I Trust You?": Explaining the Predictions of Any Classifier. Marco Tulio Ribeiro, Sameer Singh, Carlos Guestrin. KDD 2016.

[4] TreeFuser: a framework for analyzing and fusing general recursive tree traversals. Laith Sakka, Kirshanthan Sundararajah, Milind Kulkarni. OOPSLA 2017.

[5] Sound, Fine-Grained Traversal Fusion for Heterogeneous Trees. Laith Sakka, Kirshanthan Sundararajah, Ryan R. Newton, Milind Kulkarni. PLDI 2019.

[6] Parallel Schedule Synthesis for Attribute Grammars. Leo Meyerovich, Matthew Torok, Eric Atkinson, Rastislav Bodik. PPoPP 2013.

[7] A Lightweight Symbolic Virtual Machine for Solver-Aided Host Languages. Emina Torlak, Rastislav Bodik. PLDI 2014.

[8] Answering Questions about Charts and Generating Visual Explanations. Dae Hyun Kim, Enamul Hoque, Maneesh Agrawala. CHI 2020.

[9] SmBoP: Semi-autoregressive Bottom-up Semantic Parsing. Ohad Rubin, Jonathan Berant. NAACL 2021.

[10] A Syntactic Neural Model for General-Purpose Code Generation. Pengcheng Yin, Graham Neubig. ACL 2017.

[11] TaPas: Weakly Supervised Table Parsing via Pre-training. Jonathan Herzig, Paweł Krzysztof Nowak, Thomas Müller, Francesco Piccinno, Julian Martin Eisenschlos. ACL 2020.

[12] Trinity: An Extensible Synthesis Framework for Data Science. Ruben Martins, Jia Chen, Yanju Chen, Yu Feng, Isil Dillig. VLDB 2019.

[13] Maximal Multi-Layer Specification Synthesis. Yanju Chen, Ruben Martins, Yu Feng. FSE 2019.

[14] Program Synthesis Using Deduction-Guided Reinforcement Learning. Yanju Chen, Chenglong Wang, Osbert Bastani, Isil Dillig, Yu Feng. CAV 2020.

[15] Tree Traversal Synthesis Using Domain-Specific Symbolic Compilation. Yanju Chen, Junrui Liu, Yu Feng, Rastislav Bodik. ASPLOS 2022.

[16] Automated Transpilation of Imperative to Functional Code Using Neural-Guided Program Synthesis. Benjamin Mariano, Yanju Chen, Yu Feng, Greg Durrett, Isil Dillig. OOPSLA 2022.

[17] Visualization Question Answering Using Introspective Program Synthesis. Yanju Chen, Xifeng Yan, Yu Feng. PLDI 2022.