Multi-modal Program Inference:
A Marriage of Pre-trained Language Models and Component-based Synthesis

Kia Rahmani, Mohammad Raza, Sumit Gulwani, Vu Le, Daniel Morris, Arjun Radhakrishna, Gustavo Soares, Ashish Tiwari
THE STORY OF TRANSFORMERS
• BERT, ELMo and ERNIE

• Neural architectures optimized for language understanding
**PRE-TRAINED NATURAL LANGUAGE MODELS (PTM)**

- BERT, ELMo and ERNIE
- Neural architectures optimized for **language understanding**
- Trained on a large corpus of text
GPT-3 FROM OPEN AI

- Latest model from GPT-n series
- Deployed in 300 applications
  - Generates 4.5B words per day
GPT-3 FROM OPEN AI

- Latest model from GPT-n series
- Deployed in 300 applications
  - Generates 4.5B words per day
- Can be “taught” by showing a few examples of the tasks
- **Few-shot Learning**
- (Very!) diverse use-cases
GPT-3 FOR CODE GENERATION

- “Rise of AI language models in programming automation”
“Rise of AI language models in programming automation”

- Github Copilot
- A dozen programming languages
“Rise of AI language models in programming automation”

• Github Copilot
  • A dozen programming languages

• Limited Precision
FIRST HAND EXPERIMENTS WITH (NL→CODE)

- Domain of Regular Expressions (REGEX)
  - concise search patterns
  - terminals and operators

\[
\begin{align*}
i &:= \{0, 1, 2, 3, \ldots\} \\
c &:= \{A, B, \ldots, a, b, \ldots, #, $, \%, \ldots, 0, 1, 2, 3, \ldots\} \\
s &:= \text{fromChar}(c) \mid \text{range}(c, c) \mid \text{union}(s, s) \mid \text{negate}(s) \mid \text{any}() \\
e &:= \text{quant}(e, i, i) \mid \text{quantMin}(e, i) \mid \text{alter}(e, e) \mid \text{concat}(e, e) \mid \text{fromCharSet}(s)
\end{align*}
\]
• Domain of Regular Expressions (REGEX)
  • concise search patterns
  • terminals and operators

At least one digit, followed by ‘:’ at most once, followed by a digit at least zero times
FIRST HAND EXPERIMENTS WITH (NL→CODE)

- Domain of Regular Expressions (REGEX)
- concise search patterns
- terminals and operators

\[ [0-9]+:?[0-9]* \]

At least one digit, followed by ‘:’ at most once, followed by a digit at least zero times
FIRST HAND EXPERIMENTS WITH (NL→CODE)

- Domain of Regular Expressions (REGEX)
- concise search patterns
- terminals and operators

\[[0-9]+::?[0-9]\]*

At least one digit, followed by `:` at most once, followed by a digit at least zero times
First Hand Experiments with (NL→Code)

- Domain of Regular Expressions (REGEX)
- Concise search patterns
- Terminals and operators

\[ [0-9]+:?[0-9]* \]

At least one digit, followed by ‘:’ at most once, followed by a digit at least zero times.

- 2345:6789, 123
- 12, Abc
FIRST HAND EXPERIMENTS WITH (NL→CODE)

- Domain of Regular Expressions (REGEX)
- concise search patterns
- terminals and operators

\[ [0-9]+:?[0-9]\]

At least one digit, followed by ‘:’ at most once, followed by a digit at least zero times

GPT-3

2345:6789, 123
FIRST HAND EXPERIMENTS WITH (NL→CODE)

- Evaluated on 2 standard benchmark sets
- Less than 15% overall success rate
- Compared to almost 60% success rate of the state-of-the-art [2]
END OF THE STORY?
• Similarities between target and candidates:

\([0-9]+:?[0-9]^{*}\)

\[
\begin{array}{l}
( [0-9]*..:( [0-9]*)?)+ \\
( [0-9]* : [0-9]*)* \\
( [0-9]*{1, }(?: .[0-9]*){0, })* \\
[0-9]*{3} \\
( [0-9]* :)?[0-9]? \\
( digit{3})+ \\
( [0-9]* ( [:] [0-9]*)*) (0[0-9]*)+ \\
( [0-9]* .. :*[0-9]* 0*)*
\end{array}
\]
• Similarities between target and candidates:
  • Components of target are present
NOT THE END OF THE STORY!

- Similarities between target and candidates:
  - Components of target are present
  - Similar shape (operator types) to the target

```
\[
\begin{align*}
i & := \{0, 1, 2, 3, \ldots\} \\
c & := \{A, B, \ldots, a, b, \ldots, #, $, \%, \ldots, 0, 1, 2, 3, \ldots\} \\
s & := \text{fromChar}(c) \mid \text{range}(c, c) \mid \text{union}(s, s) \mid \\
& \quad \text{negate}(s) \mid \text{any}() \\
e & := \boxed{\text{quant}(e, i, i) \mid \text{quantMin}(e, i) \mid \text{alter}(e, e) \mid \\
& \quad \text{concat}(e, e) \mid \text{fromCharSet}(s)} \\
\end{align*}
\]

\[
[0-9] + \boxed{[0-9]*} \\
\]

\[
\begin{align*}
( [0-9] & \ast : ( [0-9]* \boxed{?} ) + \\
( [0-9] & \boxed{?} : [0-9] \boxed{?} ) * \\
( [0-9] \{1, \} \boxed{?} [0-9] \{0, \}) * \\
[0-9] \{3\} \\
( [0-9] & + : \boxed{?} [0-9] \boxed{?} \\
( \text{digit}(3)) + \\
( [0-9] \ast ( [\boxed{;} [0-9] \ast ] ) \ast ( 0[0-9]+ ) \\
( [0-9] \ast \ast : *[0-9] \ast 0* ) * \\
\end{align*}
\]

NLX PROGRAM SYNTHESIS FRAMEWORK

- Similarities between target and candidates:
  - Components of target are present
  - Similar shape (operator types) to the target

- NLX framework
  - Combine PTM with program synthesis
    - Handle Ambiguous Natural Language
    - Syntactically and Semantically Precisely Code
NLX PROGRAM SYNTHESIS FRAMEWORK

- NLX framework
  - Multi-modal
  - Domain agnostic

At least one digit, followed by ‘:’ at most once, followed by a digit at least zero times

GPT-3

\[
\begin{align*}
i & := \{0, 1, 2, 3, \ldots \} \\
c & := \{A, B, \ldots, a, b, \ldots, \#, $, \%, \ldots, 0, 1, 2, 3, \ldots \} \\
s & := \text{fromChar}(c) \mid \text{range}(c, c) \mid \text{union}(s, s) \mid \text{negate}(s) \mid \text{any}() \\
e & := \text{quant}(e, i) \mid \text{quantMin}(e, i) \mid \text{alter}(e, e) \mid \text{concat}(e, e) \mid \text{fromCharSet}(s)
\end{align*}
\]

[0-9]+:?[0-9]*/
COMPONENT-BASED SYNTHESIS (CBS)

- Search based approach
- Seed terms
COMPONENT-BASED SYNTHESIS (CBS)

- Search based approach
- Seed terms
- Iterative expansion
COMPONENT-BASED SYNTHESIS (CBS)

- Search based approach
- Seed terms
- Iterative expansion

\[
\begin{align*}
&\text{op}(\text{op}(t_1)) \\
&\text{op}'(\text{op}(t_1), \text{op}'(t_3, t_2)) \\
&\text{op}(t_1) \quad \text{op}(t_2) \quad \text{op}(t_3) \\
&\text{op}'(t_3, t_2) \quad \text{op}'(t_2, t_3)
\end{align*}
\]
COMPONENT-BASED SYNTHESIS (CBS)

- Search based approach
- Seed terms
- Iterative expansion
- Find consistent programs

\[ \text{op(op(t_1))} \]
\[ \text{op'(op(t_1),op'(t_3,t_2))} \]
\[ \text{op(t_1) op(t_2) op(t_3)} \]
\[ \text{op'(t_3,t_2) op'(t_2,t_3)} \]
\[ \text{...} \]

\[ \text{t_1} \quad \text{t_2} \quad \text{t_3} \]
CHALLENGES WITH CBS

- Search based approach
  - Seed terms
  - Iterative expansion
  - Find consistent programs
- Challenges:
  - Useful + concise seeds
CHALLENGES WITH SEARCH

- Search based approach
  - Seed terms
  - Iterative expansion
  - Find consistent programs

- Challenges:
  - Useful + concise seeds
  - State-space explosion
CHALLENGES WITH SEARCH

- Search based approach
  - Seed terms
  - Iterative expansion
  - Find consistent programs
- Challenges:
  - Useful + concise seeds
  - State-space explosion
  - Final ranking
NLX Solution
SEED COMPONENTS

- Extract components from PTM’s candidates

\[
( [0-9]* : ([0-9]* )? ) + \\
[0-9]* : ([0-9]* )? \\
[0-9]* : \\
([0-9]* )? \\
[0-9]* : \\
[0-9]* \\
0 \\
9 \\
. \\
. 
\]

Can become prohibitively large!
SEED COMPONENTS

- Extract components from PTM’s candidates
- Eliminate *infrequent* components

<table>
<thead>
<tr>
<th>( [0-9]* (: ([0-9]*)))?</th>
<th>( [0-9]* :)?([0-9]<em>)? ))</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>( [0-9]* : [0-9]? ))*</td>
<td>( [0-9]{3})+</td>
</tr>
<tr>
<td>( [0-9]{1, }(?: [0-9]{0, })))*</td>
<td>( [0-9]* ( [:] [0-9]<em>))</em> (0[0-9]+)</td>
</tr>
<tr>
<td>(digit){3}</td>
<td>( [0-9]* .. :<em>[0-9]</em> 0*))*</td>
</tr>
</tbody>
</table>
SEED COMPONENTS

- Extract components from PTM’s candidates
- Eliminate *infrequent* components

<table>
<thead>
<tr>
<th>Expression 1</th>
<th>Expression 2</th>
<th>Expression 3</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>[0-9]..:</code> <code>[0-9]</code>*?+</td>
<td><code>[0-9]+</code> <code>:</code>?[0-9]?</td>
<td></td>
</tr>
<tr>
<td><code>[0-9]</code>? <code>[0-9]</code>?*</td>
<td><code>[0-9]</code>{3}+</td>
<td></td>
</tr>
<tr>
<td><code>[0-9]</code>{1, } <code>(</code>? <code>:[0-9]</code>{0, }<code>)</code>*</td>
<td><code>[0-9]</code>* <code>(</code> <code>:</code> <code>[0-9]</code>*)<code>* </code>(<code>0</code>[0-9]+<code>)</code></td>
<td></td>
</tr>
<tr>
<td><code>(digit){3}</code></td>
<td><code>[0-9]</code>% <code>:</code><em>[0-9]<code>% </code>0<code>*</code></em>`</td>
<td></td>
</tr>
</tbody>
</table>
SEED COMPONENTS

- Extract components from PTM’s candidates
- Eliminate *infrequent* components
- Eliminate *redundant* components

| ( [0–9]*.. :( [0–9]*))*+ | ( [0–9]*+ :)?[0–9]?
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>( [0–9]? : [0–9]?)*</td>
<td>( [0–9]{3})+</td>
</tr>
<tr>
<td>( [0–9]{1, }?(?: .[0–9]{0, }))*)</td>
<td>( [0–9] ( [:] [0–9]<em>)</em> (0[0–9]+)</td>
</tr>
<tr>
<td>(digit){3}</td>
<td>( [0–9]* .. :<em>[0–9]</em> 0*)*</td>
</tr>
</tbody>
</table>
**SEED COMPONENTS**

- Extract components from PTM’s candidates
- Eliminate *infrequent* components
- Eliminate *redundant* components

- **Non-Maximal component:** 0, 9
- **Maximal component:** [0-9]
ITERATIVE EXPANSION

• Beam search
ITERATIVE EXPANSION

- Beam search
ITERATIVE EXPANSION

- Beam search

Only a subset of terms are considered
ITERATIVE EXPANSION

- Beam search

Only a subset of terms are considered
ITERATIVE EXPANSION

- Beam search

Only a subset of terms are considered
ITERATIVE EXPANSION

- Beam search
- Bias the search w.r.t. operator distribution
- Eliminate low-frequency operators

No need to apply \textit{Alter} at expansions

\textit{Alter} operator is NOT used

\begin{verbatim}
Alter operator is NOT used

( [0-9]*.. :([0-9]*))?
( [0-9]: [0-9])*
( [0-9]{1,})?(?: [0-9]{0,})*
[0-9]{3}
( [0-9]+ :)?[0-9]?
( digit{3})+
( [0-9]* ( [:] [0-9]*))? (0[0-9]+)
( [0-9]* .. :*[0-9]* 0*)*
\end{verbatim}
ITERATIVE EXPANSION

- Beam search
  - Bias the search w.r.t. operator distribution
  - Eliminate low-frequency operators
- How to define the beam?
• Beam search
  • Bias the search w.r.t. operator distribution
  • Eliminate low-frequency operators
• How to define the beam?
• Semantic condensation
  • Classify candidates using examples
• Beam search
  • Bias the search w.r.t. operator distribution
  • Eliminate low-frequency operators
• How to define the beam?
• Semantic condensation
  • Classify candidates using examples
  • Pick top candidates from each class
ITERATIVE EXPANSION

- Beam search
  - Bias the search w.r.t. operator distribution
  - Eliminate low-frequency operators
- How to define the beam?
- *Semantic condensation*
  - Classify candidates using examples
  - Pick top candidates from each class
A large number of programs which satisfy the examples

[0-9]+:?[0-9]*
[0-9]+:?[0-9]+
[0-9]+:?[0-3]{0,4}
[0-5]+:?[6-9]*
• A large number of programs which satisfy the examples
  • Euclidean distance
  • Levenshtein distance

\[
\begin{align*}
[0-9]+:?[0-9]^* \\
[0-9]+:?[0-9]+ \\
[0-9]+:?[0-3]{0,4} \\
[0-5]+:?[6-9]^*
\end{align*}
\]

Min (Lev + Eauc) → Final Output?
A large number of programs which satisfy the examples
- Euclidean distance
- Levenshtein distance

Min (Lev + Eauc)
EMPIRICAL RESULTS
EXPERIMENTAL EVALUATION

- Two Data sets
  - StackOverflow: 25 tasks
  - Previous work: 125 tasks
- NLX-REG outperforms the state-of-the-art
Multi-modal Program Inference: A Marriage of Pre-trained Language Models and Component-Based Synthesis

KIA RAHMANI∗, Purdue University, USA
MOHAMMAD RAZA, Microsoft, USA
SUMIT GULWANI, Microsoft, USA
VU LE, Microsoft, USA
ARJUN RADHAKRISHNA, Microsoft, USA
GUSTAVO SOARES, Microsoft, USA
ASHISH TIWARI, Microsoft, USA

Multi-modal program synthesis refers to the task of synthesizing programs from their specification given in different forms, such as a combination of natural language and examples. Examples provide a precise but incomplete specification, and natural language provides an “emptier” task description. Machine-learned pre-trained models (PTMs) are adept at handling ambiguous natural language, but struggle with generating syntactically and semantically precise code. Program synthesis techniques can generate correct code, often even from incomplete but precise specifications, such as examples, but they are unable to work with the ambiguity of natural languages. We present an approach that combines PTMs with component-based synthesis (CBS): PTMs are used to generate candidates programs from the natural language description of the task, which are then used to guide the CBS procedure to find the program that matches the precise examples-based specification. We use our combination approach to instantiate multi-modal synthesis systems for two programming domains: the domain of regular expressions and the domain of CSS selectors. Our evaluation demonstrates the effectiveness of our domain-agnostic approach in comparison to a state-of-the-art specialized system, and the generality of our approach in providing multi-modal program synthesis from natural language and examples in different programming domains.

CCS Concepts:
• Software and its engineering → Automatic programming;
• Theory of computation → Program analysis;
• Program constructs → Information extraction.

ACM Reference Format:
https://doi.org/10.1145/3485535

∗ The first author worked on this paper during an internship with the PROSE team at Microsoft.

Authors’ addresses: Kia Rahmani, Department of Computer Science, Purdue University, West Lafayette, Indiana, USA, rahmank@purdue.edu; Mohammad Raza, Microsoft, USA, moraza@microsoft.com; Sumit Gulwani, Microsoft, USA, sumitg@microsoft.com; Vu Le, Microsoft, USA, levu@microsoft.com; Daniel Morris, Microsoft, USA, Daniel.Morris@microsoft.com; Arjun Radhakrishna, Microsoft, USA, arradha@microsoft.com; Gustavo Soares, Microsoft, USA, Gustavo.Soares@microsoft.com; Ashish Tiwari, Microsoft, USA, astiwar@microsoft.com.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
© 2021 Copyright held by the owner/author(s).
2475-1421/2021/10-ART158
https://doi.org/10.1145/3485535

READ MORE!

- Ablation Study
- Domain of CSS selector
- Optimized use of the PTM
RECAP

- PTM: “Rise of AI Language Models in Programming Automation”
- Multi-modal -> precision
- NLX: component-based synthesis based on results generated from a PTM
  - Domain Agnostic (REGEX and CSS selectors)
- Other domains + general purpose programming
ACKNOWLEDGMENT

Mohammad Raza (moraza@microsoft.com)
Sumit Gulwani (sumitg@microsoft.com)
Ashish Tiwari (astiwar@microsoft.com)
Gustavo Soares (Gustavo.Soares@microsoft.com)
Arjun Radhakrishna (arradha@microsoft.com)
Daniel Morris (Daniel.Morris@microsoft.com)
Vu Le (levu@microsoft.com)
THANKS FOR YOUR ATTENTION!