Hoole+: Program Synthesis by Type-Guided Abstraction Refinement

Zheng Guo, Michael James, David Justo, Jiaxiao Zhou, Ziteng Wang, Ranjit Jhala, Nadia Polikarpova
University of California San Diego
Outline

01 Problem: Component-Based Synthesis
   Running example
   Previous solution: SyPet

02 Challenge: Polymorphism
   Search space explosion

03 Solution: Type-Guided Abstraction Refinement
   Abstraction
   Refinement
   Evaluation

04 Hoogle+: More Features
Outline

01 Problem: Component-Based Synthesis

Running example
Previous solution: SyPet

02 Challenge: Polymorphism

Search space explosion

03 Solution: Type-Guided Abstraction Refinement

Abstraction
Refinement
Evaluation

04 Hoogle+: More Features
### Component-Based Synthesis

#### Example

<table>
<thead>
<tr>
<th>Default value</th>
<th>List of optional values</th>
<th>Desired result</th>
</tr>
</thead>
<tbody>
<tr>
<td>'a'</td>
<td>[Nothing, Just 'b', Nothing, Just 'c', Just 'd']</td>
<td>'b'</td>
</tr>
<tr>
<td>0</td>
<td>[Nothing, Nothing, Nothing, Nothing]</td>
<td>0</td>
</tr>
</tbody>
</table>

```
d: a
xs: [Maybe a]
a
```
## Component-Based Synthesis

### Example

<table>
<thead>
<tr>
<th>Default value</th>
<th>List of optional values</th>
<th>Desired result</th>
</tr>
</thead>
<tbody>
<tr>
<td>'a'</td>
<td>[Nothing, Just 'b', Nothing, Just 'c', Just 'd']</td>
<td>'b'</td>
</tr>
<tr>
<td>0</td>
<td>[Nothing, Nothing, Nothing, Nothing]</td>
<td>0</td>
</tr>
</tbody>
</table>

\[
d: a \rightarrow xs: [Maybe a] \rightarrow a
\]
Component-Based Synthesis

Example

set:haskell-platform

package base
- Basic libraries. This package contains the Standard Haskell Prelude and its support libraries, and a large collection of useful functions.

package bytestring
- Fast, compact, strict and lazy byte strings with a list interface. An efficient compact, immutable byte string type (both strict and lazy), with a powerful API and many useful features.

package containers
- Assorted concrete container types. This package contains efficient general-purpose implementations of various immutable container types. The package provides with examples of common operations see the containers introduction. The declared cost of each operation

package text
- An efficient packed Unicode text type. An efficient packed, immutable Unicode text type (both strict and lazy), with a powerful API and some efficient operators. This package provides text manipulation capabilities that are optimized for performance.
Solution: \( d \) \( xs \rightarrow \) \textbf{fromMaybe} \( d \) \((\textbf{listToMaybe} (\textbf{catMaybes} \( xs \)))\)

Component-Based Synthesis

Example

\[
[ \text{Nothing}, \text{Just 'b'}, \text{Nothing}, \text{Just 'c'}, \text{Just 'd'}}] \\
\rightarrow \ [ 'b', 'c', 'd' ]
\]
Example

Component-Based Synthesis

Solution: \( d \to \text{fromMaybe} \ d \ (\text{listToMaybe} \ (\text{catMaybes} \ xs)) \)
01 Problem: Component-Based Synthesis
   Running example
   Previous solution: SyPet

02 Challenge: Polymorphism
   Search space explosion

03 Solution: Type-Guided Abstraction Refinement
   Abstraction
   Refinement
   Evaluation

04 Hoogle+: More Features
Component-Based Synthesis

Synthesis to the rescue

Type Query

Components

Synthesizer

Result Program
fromMaybe :: a -> Maybe a -> a
fromMaybe :: a -> Maybe a -> a
fromMaybe :: a → Maybe a → a
Previous Solution
Petri net-Based Search

Type Query
\[d: a \rightarrow \text{xs: [Maybe a]} \rightarrow a\]
Previous Solution

Petri net-Based Search

Type Query
$$d: a \rightarrow xs: [Maybe a] \rightarrow a$$

Feng et al. POPL ‘17
Previous Solution

Petri net-Based Search

Type Query

d: a -> xs: [Maybe a] -> a

Feng et al. POPL ’17

<table>
<thead>
<tr>
<th>catMaybes xs</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
</tr>
<tr>
<td>head</td>
</tr>
<tr>
<td>[a]</td>
</tr>
<tr>
<td>fromMaybe</td>
</tr>
<tr>
<td>listToMaybe</td>
</tr>
<tr>
<td>catMaybes</td>
</tr>
</tbody>
</table>

Maybe a

| head         |
| [Maybe a]    |
Previous Solution
Petri net-Based Search

Feng et al. POPL ’17

Type Query
d: a -> xs: [Maybe a] -> a

listToMaybe (catMaybes xs)
SOLUTION: \( \text{fromMaybe} \ d \ (\text{listToMaybe} \ (\text{catMaybes} \ xs)) \)
01 Problem: Component-Based Synthesis
- Running example
- Previous solution: SyPet

02 Challenge: Polymorphism
- Search space explosion

03 Solution: Type-Guided Abstraction Refinement
- Abstraction
- Refinement
- Evaluation

04 Hoogle+: More Features
Challenge

Polymorphic components

$\text{fromMaybe :: } a \rightarrow \text{Maybe } a \rightarrow a$

$\forall \alpha. \alpha \rightarrow \text{Maybe } \alpha \rightarrow \alpha$
Polymorphic components

fromMaybe :: \( \forall \alpha. \alpha \rightarrow \text{Maybe } \alpha \rightarrow \alpha \)

- **a**
  - fromMaybe
    - fromMaybe
      - fromMaybe
        - fromMaybe
          - fromMaybe
            - Maybe (Maybe (Maybe a))
          - Maybe (Maybe a)
    - Maybe a
      - fromMaybe
        - fromMaybe
          - fromMaybe
            - Maybe a
Challenge

Polymorphic components

- `fromMaybe
- `head
- `listToMaybe
- `Maybe a
- `[a]

- `catMaybes
- `listToMaybe
- `[[a]]

- `Maybe (Maybe a)
- `Maybe [a]
- `Maybe [[a]]

- `Maybe (Maybe (Maybe a))

- `...
**Challenge**

**Polymorphic components**

---

Infinite graph!
Outline

01 Problem: Component-Based Synthesis
   Running example
   Previous solution: SyPet

02 Challenge: Polymorphism
   Search space explosion

03 Solution: Type-Guided Abstraction Refinement
   Abstraction
   Refinement
   Evaluation

04 Hoogle+: More Features
Type-Guided Abstraction Refinement

Abstract types

\( \tau \)

[\([\tau]\)]

[\([a]\)]

[\([[[a]]]\)]

Maybe \( \tau \)

Maybe [a]

Maybe (Maybe a)
Type-Guided Abstraction Refinement

Abstract petri net
Type-Guided Abstraction Refinement

Abstract petri net

```
fromMaybe

τ

head

listToMaybe

Maybe (Maybe a)

fromMaybe

head

listToMaybe

Maybe (Maybe (Maybe a))

listToMaybe

Maybe [a]

listToMaybe

Maybe [[a]]
```

```
head

[a]

listToMaybe

Maybe (Maybe a)
```
Type-Guided Abstraction Refinement

Abstract petri net

d: a -> xs: [Maybe a] -> a
Type-Guided Abstraction Refinement

Abstract petri net

SOLUTION: \( \text{\texttt{d}} \text{ \textit{xs}} \rightarrow \text{\texttt{fromMaybe d (listToMaybe (catMaybes \text{\textit{xs}}))}} \)
SOLUTION: \( \lambda d \, xs \to \text{fromMaybe} \, d \,(\text{listToMaybe} \,(\text{catMaybes} \, xs)) \)
**Type-Guided Abstraction Refinement**

**Abstract petri net**

**SOLUTION:** \( \lambda d \, xs \to \text{fromMaybe} \, d \, (\text{listToMaybe} \, (\text{catMaybes} \, xs)) \)
FromMaybe

**SOLUTION**: \( \lambda d \, xs \to \textbf{fromMaybe} \; d \; (\textbf{listToMaybe} \; (\textbf{catMaybes} \; xs)) \)
Type-Guided Abstraction Refinement

Abstract petri net

Type Query
d: a -> xs: [Maybe a] -> a
Type-Guided Abstraction Refinement

Abstract Petri Net

**SOLUTION**: \( \text{\textbackslash d \text{x}s} \rightarrow \text{fromMaybe \text{\textbackslash d} (\text{catMaybes \text{x}s})} \)

Type Query

\( \text{d: a} \rightarrow \text{xs: [Maybe a]} \rightarrow \text{a} \)
**Type-Guided Abstraction Refinement**

Abstract petri net

Type Query

\[ d: a \rightarrow xs: [Maybe a] \rightarrow a \]

Solution: \ \[ \texttt{fromMaybe } d \ (\texttt{catMaybes } xs) \]
Spurious Program: \( \backslash d \; xs \rightarrow \text{fromMaybe} \; d \; (\text{catMaybes} \; xs) \)

AST of the program

Type checking of the program

d: a \rightarrow xs: [Maybe a] \rightarrow a

\text{catMaybes} :: \forall \alpha. [Maybe \alpha] \rightarrow [\alpha]

xs :: [Maybe a]

catMaybes \; xs :: [a]
Type Query

\[ d: a \rightarrow xs: [Maybe a] \rightarrow a \]

Abstract Petri net

Type-Guided Abstraction Refinement

Spurious Program: \( \lambda d \; xs \rightarrow \text{fromMaybe} \; d \; (\text{catMaybes} \; xs) \)

Type Error!

fromMaybe :: \( \forall \alpha. \alpha \rightarrow \text{Maybe} \; \alpha \rightarrow \alpha \)
Type-Guided Abstraction Refinement

TyGAR Workflow

- Initial Abstraction
- Type Checker
- Abstract Reachability
- Refinement
- Check OK
- Result Program
- Query Type
- Components
- Path
- No Path
- New Abstraction
- Type Error
01 Problem: Component-Based Synthesis
   Running example
   Previous solution: SyPet

02 Challenge: Polymorphism
   Search space explosion

03 Solution: Type-Guided Abstraction Refinement
   Abstraction
   Refinement
   Evaluation

04 Hoogle+: More Features
Type Query
d: a -> xs: [Maybe a] -> a

Spurious Program: \d xs -> fromMaybe d (catMaybes xs)

Type-Guided Abstraction Refinement
Type abstraction refinement

fromMaybe

d
catMaybes

\[
\begin{array}{c}
\text{fromMaybe} \\
d \\
catMaybes
\end{array}
\]

\[
\begin{array}{c}
x \rightarrow \\
a \\
[\text{Maybe a}]
\end{array}
\]

fromMaybe :: \forall \alpha. \alpha -> \text{Maybe } \alpha -> \alpha

\[
\begin{array}{c}
\text{fromMaybe} \\
\tau
\end{array}
\]

\[
\begin{array}{c}
[\tau] \\
[a] \\
\text{Maybe a}
\end{array}
\]

AST of the program  Type checking of the program
Type-Guided Abstraction Refinement

Refined abstract petri net

Type Query
\( d : a \rightarrow xs : \text{[Maybe a]} \rightarrow a \)

Spurious Program:
\( \lambda d \; xs \rightarrow \text{fromMaybe} \; d \;(\text{catMaybes} \; xs) \)
Type-Guided Abstraction Refinement

Refined abstract petri net

Type Query
\[ d : a \rightarrow xs : [Maybe a] \rightarrow a \]

Spurious Program:
\[ \lambda d \hspace{1mm} xs \rightarrow \text{fromMaybe} \hspace{1mm} d \hspace{1mm} (\text{catMaybes} \hspace{1mm} xs) \]
Type-Guided Abstraction Refinement

Refined abstract petri net

Type Query
\(d: a \rightarrow xs: [\text{Maybe} \ a] \rightarrow a\)

Spurious Program:
\(\lambda d \ xs \rightarrow \text{fromMaybe} \ d \ (\text{catMaybes} \ xs)\)
Type-Guided Abstraction Refinement

Refined abstract petri net

Type Query
\[ d: a \rightarrow \text{xs: [Maybe a]} \rightarrow a \]

Solution:
\[ \lambda d \ xs \rightarrow \text{fromMaybe} d (\text{listToMaybe} (\text{catMaybes} \ xs)) \]
Type-Guided Abstraction Refinement

Refined abstract petri net

Type Query

d: a -> xs: [Maybe a] -> a

\[d \ xs \rightarrow \text{fromMaybe} \; d \; (\text{listToMaybe} \; (\text{catMaybes} \; \xs))\]
Type-Guided Abstraction Refinement
Refined abstract petri net

Type Query
\[ d : a -> xs : [\text{Maybe } a] -> a \]

Solution:
\[ d \text{ xs} -> \text{fromMaybe } d (\text{listToMaybe } (\text{catMaybes } \text{xs})) \]
Refined abstract petri net

Type-Guided Abstraction Refinement

Solution:
\[ d \, xs \rightarrow \text{fromMaybe} \, d \,(\text{listToMaybe} \,(\text{catMaybes} \, xs)) \]
Outline

01 Problem: Component-Based Synthesis
   Running example
   Previous solution: SyPet

02 Challenge: Polymorphism
   Search space explosion

03 Solution: Type-Guided Abstraction Refinement
   Abstraction
   Refinement
   Evaluation

04 Hoogle+: More Features
Type-Guided Abstraction Refinement

Evaluation

Components

291 components
12 popular Haskell library modules

Benchmarks

24 benchmarks from Hoogle
6 benchmarks from StackOverflow
14 benchmarks curated by us
Type-Guided Abstraction Refinement Evaluation

# of benchmarks

Too many refinements ➔ Too large petri net

Abstraction w/ Refinement
Type-Guided Abstraction Refinement

Evaluation

<table>
<thead>
<tr>
<th># of benchmarks</th>
<th>Abstraction w/ Refinement</th>
<th>Abstraction w/o Refinement</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>34</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>37</td>
<td></td>
</tr>
</tbody>
</table>

Too many refinements ➔ Too large petri net
No refinement ➔ Poor at hard queries

Query: f: (a -> b) -> g: (a -> c) -> x: a -> (b, c)
Solution: \( f \ g \ x \rightarrow (f \ x, g \ x) \)
**Type-Guided Abstraction Refinement**

**Evaluation**

- **Too many refinements** → **Too large petri net**
- **No refinement** → **Poor at hard queries**

**Query:** \( f: (a \rightarrow b) \rightarrow g: (a \rightarrow c) \rightarrow x: a \rightarrow (b, c) \)**

**Solution:** \( \langle f \ g \ x \rightarrow (f \ x, g \ x) \rangle \)

Bounded refinements 2s!
Outline

01 Problem: Component-Based Synthesis
   Running example
   Previous solution: SyPet

02 Challenge: Polymorphism
   Search space explosion

03 Solution: Type-Guided Abstraction Refinement
   Abstraction
   Refinement
   Evaluation

04 Hoogle+: More Features
More advanced Haskell features

higher-order functions, type classes, etc.

Filter out uninteresting solutions

e.g. \( d \ \text{xs} \rightarrow \text{fromLeft} \ d \ (\text{Right} \ \text{xs}) \) always returns \( d \)
Outline

01 Problem: Component-Based Synthesis
   Running example
   Previous solution: SyPet

02 Challenge: Polymorphism
   Search space explosion

03 Solution: Type-Guided Abstraction Refinement
   Abstraction
   Refinement
   Evaluation

04 Hoogle+: More Features