Resource-Guided Program Synthesis

Tristan Knoth\textsuperscript{1}, Di Wang\textsuperscript{2}, Nadia Polikarpova\textsuperscript{1}, Jan Hoffmann\textsuperscript{2}

\textsuperscript{1}UC San Diego
\textsuperscript{2}Carnegie Mellon University

PLDI 2019
Program Synthesis

Declarative specification

Synthesizer

Executable program
State of the art

“Find the intersection of two sorted lists”
Type-directed synthesis

\[ \text{common} :: \text{xs: SList a} \to \text{ys: SList a} \]
\[ \to \nu: \{\text{List a} \mid \text{elems } \nu = \text{elems } \text{xs} \cap \text{elems } \text{ys}\} \]

\[
\text{common} = \lambda \text{xs}. \lambda \text{ys}.
\begin{align*}
\text{match } \text{xs with} \\
\text{Nil } &\to \text{Nil} \\
\text{Cons } x \text{ xt } &\to \\
&\text{if } !(\text{member } x \text{ ys}) \\
&\text{then } \text{common } \text{xt } \text{ys} \\
&\text{else } \text{Cons } x (\text{common } \text{xt } \text{ys})
\end{align*}
\]
common = λ xs. λ ys.
match xs with
  Nil → Nil
  Cons x xt →
    if !(member x ys)
    then common xt ys
    else Cons x (common xt ys)
common = \lambda \, xs. \, \lambda \, ys. \\
match \, xs \, with \\
\quad Nil \rightarrow Nil \\
\quad Cons \, x \, xt \rightarrow \\
\quad \quad if \, !(member \, x \, ys) \\
\quad \quad \quad then \, common \, xt \, ys \\
\quad \quad else \, Cons \, x \, (common \, xt \, ys)
common = \lambda \; xs. \; \lambda \; ys. \\
match \; xs \; \text{with} \\
\text{Nil} \rightarrow \text{Nil} \\
\text{Cons} \; x \; xt \rightarrow \\
match \; ys \; \text{with} \\
\text{Nil} \rightarrow \text{Nil} \\
\text{Cons} \; y \; yt \rightarrow \\
\text{if} \; x < y \\
\text{then} \; \text{common} \; xt \; ys \\
\text{else if} \; y < x \\
\text{then} \; \text{common} \; xs \; yt \\
\text{else} \; \text{Cons} \; x \; (\text{common} \; xs \; ys)
common = \lambda \, \text{xs}. \lambda \, \text{ys}. \, \\
\text{match } \text{xs with} \\
\quad \text{Nil } \rightarrow \ \text{Nil} \\
\quad \text{Cons x xt } \rightarrow \\
\hspace{1em} \text{match } \text{ys with} \\
\hspace{2em} \text{Nil } \rightarrow \ \text{Nil} \\
\hspace{2em} \text{Cons y yt } \rightarrow \\
\hspace{3em} \text{if } x < y \\
\hspace{4em} \text{then common xt ys} \\
\hspace{4em} \text{else if } y < x \\
\hspace{5em} \text{then common xs yt} \\
\hspace{4em} \text{else Cons x (common xs ys)}
common = \lambda xs. \lambda ys.
  match xs with
    Nil → Nil
    Cons x xt →
      if !(member x ys)
        then common xt ys
        else Cons x (common xt ys)

O(m \cdot n)

common = \lambda xs. \lambda ys.
  match xs with
    Nil → Nil
    Cons x xt →
      match ys with
        Nil → Nil
        Cons y yt →
          if x < y
            then common xt ys
            else if y < x
              then common xs yt
              else Cons x (common xs ys)

O(m + n)
What we have

“Find the intersection of two sorted lists”

What we want

“Find the intersection of two sorted lists in linear time”

Synthesizer

$O(m \cdot n)$

$O(m+n)$
ReSyn

The first resource-aware synthesizer for recursive programs
This talk

1. Specification

“Find the intersection of two sorted lists in linear time”
This talk

1. Specification

“Find the intersection of two sorted lists in linear time”

Synthesizer
This talk

1. Specification

“Find the intersection of two sorted lists in linear time”
This talk

1. Specification
2. Analysis

“Find the intersection of two sorted lists in linear time”
This talk

1. Specification
2. Analysis
3. Search

“Find the intersection of two sorted lists in linear time”
This talk

1. Specification
2. Analysis
3. Search
“Find the intersection of two sorted lists in linear time”
“Find the intersection of two sorted lists in linear time”

Refinement types

Synthesizer

??
“Find the intersection of two sorted lists in linear time”

Refinement types with Resource annotations
Refinements:
Synquid

Type-directed Program Synthesis

Resource-guided Program Synthesis

[Polikarpova et. al 2016]
Refinements: Synquid

Resource annotations:
Automated Amortized Resource Analysis

Type-directed Program Synthesis

Resource-guided Program Synthesis

[Polikarpova et al. 2016]  [Hoffmann et al. 2010]
“Find the intersection of two sorted lists in linear time”
v : { \text{Int} \mid v \geq 0 \}
Refinement types

common = ??
Refinement types

common :: xs: SList a → ys: SList a
       → ν: {List a | elems ν = elems xs ∩ elems ys}
common = ??
Refinement types

common :: xs: SList a → ys: SList a → v: {List a | elems v = elems xs ∩ elems ys}
common = ??
Refinement types

common :: xs: SList a → ys: SList a
        → v: {List a | elems v = elems xs ∩ elems ys}
common = ??
Functional specification

Synquid

Library functions

[Polikarpova et. al, 2016]
Functional specification

Common = \lambda xs. \lambda ys.
match xs with
Nil → Nil
Cons x xt →
if !(member x ys)
then common xt ys
else Cons x (common xt ys)

Library functions

[Polikarpova et. al, 2016]
“Find the intersection of two sorted lists in linear time”
“Find the intersection of two sorted lists in linear time”
“Find the intersection of two sorted lists in linear time”

Potential: numeric

Refinement: boolean
Resource annotations

common :: xs: SList a → ys: SList a
    → v: {List a | elems v = elems xs ∩ elems ys}
common = ??
Resource budget

common :: xs: SList a → ys: SList a
        → v: {List a | elems v = elems xs ∩ elems ys}
common = ??
Synthesize with ReSyn

\[
\text{common} :: \text{xs}: \text{SList } a \rightarrow \text{ys}: \text{SList } a \\
\quad \rightarrow \nu: \{\text{List } a | \text{elems } \nu = \text{elems } \text{xs} \cap \text{elems } \text{ys}\}
\]
\[
\text{common} = ??
\]
Components: \texttt{member}

\[
\text{member :: } z:a \rightarrow zs: \text{List } a \rightarrow v:\{\text{Bool}|v = (x \in \text{elems } xs)\}
\]
Components: member

member :: z:a → zs: List a₁
→ v:{Bool|v = (x \in \text{elems} \ xs)}
Components: \texttt{member}

\texttt{member :: z:a \rightarrow zs: List a^1}
\rightarrow \nu:\{\text{Bool}|\nu = (x \in \text{elems } xs)\}
Functional specification

Resource bound

Library functions

ReSyn
Functional specification

Resource bound

Library functions

reSyn

common = \( \lambda xs. \lambda ys. \)

match xs with
  Nil -> Nil
  Cons x xt ->

match ys with
  Nil -> Nil
  Cons y yt ->

if x < y then common xt ys
else
  if y < x then common xs yt
  else Cons x (common xs ys)
This talk

1. Specification
2. Analysis
3. Search
How do we know common does not run in linear time?

\[
\text{common} = \lambda \text{xs. } \lambda \text{ys.}
\]
\[
\text{match xs with}
\]
\[
\text{Nil} \to \text{Nil}
\]
\[
\text{Cons x xt} \to
\]
\[
\text{if } !\text{(member x ys)}
\]
\[
\text{then common xt ys}
\]
\[
\text{else Cons x (common xt ys)}
\]
common = λ xs. λ ys.
  match xs with
    Nil → Nil
    Cons x xt →
      if !(member x ys)
      then common xt ys
      else Cons x (common x ys)

member :: z:a → zs: List a¹
      → ν:{Bool|ν = (x ∈ elems xs)}
How do we automate this reasoning?

\[
\text{common} = \lambda \text{xs}. \lambda \text{ys}. \\
\text{match} \ \text{xs} \ \text{with} \\
\quad \text{Nil} \rightarrow \text{Nil} \\
\quad \text{Cons} \ x \ \text{xt} \rightarrow \\
\quad \quad \text{if} \ !\!(\text{member} \ x \ \text{ys}) \\
\quad \quad \ \text{then} \ \text{common} \ \text{xt} \ \text{ys} \\
\quad \quad \ \text{else} \ \text{Cons} \ x \ (\text{common} \ \text{xt} \ \text{ys})
\]
common :: xs: SList a → ys: SList a → v: {List a |…} common = λ xs. λ ys.
  match xs with
  Nil → Nil
  Cons x xt →
    if !(member x ys)
      then common xt ys
    else Cons x (common xt ys)
Can we partition the allotted resources between all function calls?

\[
\text{common} = \lambda \, \text{xs}. \lambda \, \text{ys}. \quad \text{ys} \:: \text{SList} \, \text{a}^1
\]

\[
\text{match} \, \text{xs} \, \text{with} \\
\text{Nil} \rightarrow \text{Nil} \\
\text{Cons} \, x \, \text{xt} \rightarrow \\
\quad \text{if} \! \left(\neg \text{member} \, x \, \text{ys}\right) \\
\quad \text{then} \, \text{common} \, \text{xt} \, \text{ys} \\
\quad \text{else} \, \text{Cons} \, x \, (\text{common} \, \text{xt} \, \text{ys})
\]
common = \lambda xs. \lambda ys.

match xs with
  Nil \rightarrow Nil
  Cons x xt \rightarrow
    if !(member x ys)
    then common xt ys
    else Cons x (common xt ys)
common = λ xs. λ ys.

match xs with
    Nil → Nil
    Cons x xt →
        if !(member x (ys :: List a))
            then common xt (ys :: List a)
            else Cons x (common xt ys)
\begin{verbatim}
common = \lambda \text{xs}. \lambda \text{ys}.
\hspace{1em}
match \text{xs} with
\hspace{1em}
\hspace{1em}Nil \rightarrow \text{Nil}
\hspace{1em}
\hspace{1em}\text{Cons} \ x \ \text{xt} \rightarrow
\hspace{1em}
\hspace{2em}if !(\text{member} \ x \ (\text{ys} :: \text{List} \ \text{a}^p))
\hspace{2em}then \ \text{common} \ \text{xt} \ \text{ys}
\hspace{2em}else \ \text{Cons} \ x \ (\text{common} \ \text{xt} \ \text{ys})
\end{verbatim}
member :: z:a → zs: List a¹ → v:{Bool|...}

common = λ xs. λ ys.
  match xs with
  Nil → Nil
  Cons x xt →
    if !(member x (ys :: List aᵖ))
      then common xt ys
    else Cons x (common xt ys)
member :: z:a → zs: List a¹ → v:{Bool|...}

common = λ xs. λ ys. match xs with
       Nil → Nil
     Cons x xt →
       if !(member x (ys :: List a^p))
         then common xt ys
       else Cons x (common xt ys)
\[
\begin{align*}
    a & < : b \quad p \geq q \\
    \hline
    a^p & < : b^q \\
    \hline
    \text{List } a^p & < : \text{List } b^q
\end{align*}
\]
member :: z:a → zs: List a¹ → v:{Bool|…}

common = λ xs. λ ys.
  match xs with
    Nil → Nil
  Cons x xt →
    if !(member x (ys :: List aᵖ))
      then common xt ys
      else Cons x (common xt ys)
common :: xs: SList a₁ → ys: SList a₁ → v: {List a |…}

common = λ xs. λ ys.
  match xs with
  Nil → Nil
  Cons x xt →
    if !(member x ys)
      then common xt (ys :: List a_q)
      else Cons x (common xt ys)
common :: xs: SList a↑ → ys: SList a↑ → ν: {List a |...}

common = λ xs. λ ys.
    match xs with
    Nil → Nil
    Cons x xt →
        if !(member x ys)
            then common xt (ys :: List a^q)
            else Cons x (common xt ys)

List a^q <: List a↑
    q ≥ 1
Sharing $\rightarrow$ \text{SList} a^1 $\triangleright$ \text{SList} a^p, \text{SList} a^q

\text{common} = \lambda \text{xs}. \lambda \text{ys}.
\text{match} \text{xs} \text{ with }
\text{Nil} \rightarrow \text{Nil}
\text{Cons} x \text{ xt} \rightarrow
\text{if} !(\text{member} x \text{ ys})
\text{then} \text{common} \text{ xt} \text{ ys}
\text{else} \text{Cons} x (\text{common} \text{ xt} \text{ ys})
Sharing $\rightarrow$ SList $a^1$ $\Downarrow$ SList $a^p$, SList $a^q$

common = $\lambda$ xs. $\lambda$ ys.

match xs with
  Nil $\rightarrow$ Nil
  Cons x xt $\rightarrow$
    if !(member x ys)
    then common xt ys
    else Cons x (common xt ys)

1 = p + q
common = λ xs. λ ys.
  match xs with
   Nil → Nil
  Cons x xt →
    if !(member x ys) then common xt ys else Cons x (common xt ys)
1 = p + q &&
   p ≥ 1 &&
   q ≥ 1

SMT
SMT

\[
1 = p + q \quad && \\
p \geq 1 \quad && \\
q \geq 1
\]
This talk

1. Specification
2. Analysis
3. Search
Enumerate-and-check

Synthesizer → Whole-program Resource Analysis
Enumerate-and-check

Resource-Guided Synthesis

Synthesizer → Whole-program Resource Analysis → Enumerated programs → Synthesis → Local Resource Analysis → Check

Synthesizer → Whole-program Resource Analysis → Enumerated programs → Synthesis → Local Resource Analysis → Check
Reject impossible programs early

common = \lambda xs. \lambda ys. 
\textbf{match} xs \textbf{with} 
\hspace{1em} \text{Nil} \rightarrow \text{Nil} 
\hspace{1em} \text{Cons} x xt \rightarrow 
\hspace{1em} \hspace{1em} \textbf{if} !\text{(member} x ys) 
\hspace{1em} \hspace{1em} \hspace{1em} \textbf{then} common xt ys 
\hspace{1em} \hspace{1em} \textbf{else} ??
Reject impossible programs early with local analysis

\[ \text{common} = \lambda \text{xs. } \lambda \text{ys.} \]
\[
\text{match xs with}
\]
\[
\text{Nil} \rightarrow \text{Nil}
\]
\[
\text{Cons x xt} \rightarrow
\]
\[
\text{if } ! (\text{member x ys})
\]
\[
\text{then common xt ys}
\]
\[
\text{else } ??
\]
Reject impossible programs early with local analysis

\[
\text{common} = \lambda \text{xs}. \lambda \text{ys}. \\
\text{match } \text{xs} \text{ with} \\
\text{Nil } \rightarrow \text{ Nil} \\
\text{Cons } x \text{ xst } \rightarrow \\
\text{if } !\left(\text{member } x \text{ ys}\right) \text{ then} \text{ common } xst \text{ ys} \\
\text{else } ?? \\
\]
Reject impossible programs early with local analysis

\[ \text{common} = \lambda \text{xs. } \lambda \text{ys.} \]

\[ \text{match xs with} \]
\[ \text{Nil } \rightarrow \text{Nil} \]
\[ \text{Cons } x \text{ xt } \rightarrow \]
\[ \text{if } ! (\text{member x ys}) \]
\[ \text{then common ys ??} \]
\[ \text{else ??} \]
Evaluation
Evaluation

1. Can ReSyn generate faster programs than Synquid?
Evaluation

1. Can ReSyn generate faster programs than Synquid?
2. How much longer does ReSyn take to generate code?
Evaluation

1. Can ReSyn generate faster programs than Synquid?
2. How much longer does ReSyn take to generate code?
3. Is local resource analysis effective at guiding the search?
1. Can ReSyn generate faster programs?
1. Can ReSyn generate faster programs?

- Generated by Synquid
- Generated by ReSyn

Require super-linear bound
1. Can ReSyn generate faster programs?

- Generated by Synquid
- Generated by ReSyn

Require super-linear bound

Improved by ReSyn
1. Can ReSyn generate faster programs?

- Generated by ReSyn

- Improved by ReSyn
**compress:** Remove adjacent duplicates

\[
\text{compress } xs =
\begin{align*}
&\text{match } xs \text{ with} \\
&\text{Nil } \rightarrow \text{ Nil} \\
&\text{Cons } x3 \ x4 \rightarrow \\
&\quad \text{match } \text{compress } x4 \text{ with} \\
&\quad \text{Nil } \rightarrow \text{ Cons } x3 \ 	ext{Nil} \\
&\quad \text{Cons } x10 \ x11 \rightarrow \\
&\quad \quad \text{if } x3 == x10 \\
&\quad \quad \quad \text{then compress } x4 \\
&\quad \quad \quad \text{else Cons } x3 \ \text{(Cons } x10 \ x11) \\
\end{align*}
\]

\[O(2^n)\]  
Synquid

\[O(n)\]  
ReSyn
**insert**: Insert into a sorted list

\[
\text{insert } x \ x s = \\
\text{match } x s \text{ with} \\
\quad \text{Nil } \rightarrow \text{ Cons } x \ \text{Nil} \\
\quad \text{Cons } y \ ys \rightarrow \\
\quad \quad \text{if } x < y \\
\quad \quad \quad \text{then } \text{ Cons } x \ (\text{insert } y \ ys) \\
\quad \quad \quad \text{else } \text{ Cons } y \ (\text{insert } x \ ys)
\]

\[
\text{O(n)} \\
\text{Synquid}
\]

\[
\text{O(n)} \\
\text{ReSyn}
\]
insert :: x:a → xs: SList a → v:{SList a | elems v = elems xs ∪ {x}}

insert x xs =
  match xs with
  Nil → Cons x Nil
  Cons y ys →
    if x < y
    then Cons x (insert y ys)
    else Cons y (insert x ys)

“One recursive call per element in xs that is smaller than x”
2. How do synthesis times compare?
2. How do synthesis times compare?

Median: 2.5x slower
2. How do synthesis times compare?

ReSyn finds faster implementation
2. How do synthesis times compare?
3. Does local resource analysis guide synthesis?
3. What happens if the analysis is non-local?
3. What happens if the analysis is non-local?

- ReSyn
- Enumerate-and-check

Timeout > 600s
3. What happens if the analysis is non-local?

```
common = λ xs. λ ys.
match xs with
  Nil → Nil
  Cons x xt →
    if !(member x ys)
      then common xt ys
      else Cons x (common x ys)
```

Timeout > 600s
What we had

“Find the intersection of two sorted lists”

\[ O(m \cdot n) \]

What we have now

“Find the intersection of two sorted lists in linear time”

\[ O(m+n) \]
https://bitbucket.org/tjknoth/resyn