Resource-Guided Program Synthesis

Tristan Knoth$^1$, Di Wang$^2$, Nadia Polikarpova$^1$, Jan Hoffmann$^2$

$^1$UC San Diego

$^2$Carnegie Mellon University

PLDI 2019
Program Synthesis

Declarative specification

Synthesizer

Executable program
State of the art

“Find the intersection of two sorted lists”

Synthesizer
Type-directed synthesis

```plaintext
common :: xs: SList a → ys: SList a
          → ν: {List a | elems ν = elems xs ∩ elems 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 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 xt 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 = \lambda\, xs. \lambda\, ys.

\text{match } xs \text{ with}
\hspace{1em} \text{Nil } \rightarrow \text{ Nil}
\hspace{1em} \text{Cons } x\, xt \rightarrow
\hspace{2em} \text{match } ys \text{ with}
\hspace{3em} \text{Nil } \rightarrow \text{ Nil}
\hspace{3em} \text{Cons } y\, yt \rightarrow
\hspace{4em} \text{if } x < y
\hspace{5em} \text{then } \text{common } xt\, ys
\hspace{5em} \text{else if } y < x
\hspace{6em} \text{then } \text{common } xs\, yt
\hspace{6em} \text{else } \text{Cons } x\, (\text{common } xs\, ys)
common = \lambda \text{xs}. \lambda \text{ys}. \\
\text{match \, \text{xs} \, with} \\
\text{Nil} \to \text{Nil} \\
\text{Cons \, x \, \text{xt} \to} \\
\text{match \, \text{ys} \, with} \\
\text{Nil} \to \text{Nil} \\
\text{Cons \, y \, \text{yt} \to} \\
\text{if \, x < y} \\
\text{then \, \text{common \, xt \, ys}} \\
\text{else if \, y < x} \\
\text{then \, \text{common \, xs \, yt}} \\
\text{else \, Cons \, x \, (\text{common \, xs \, ys})}
common = \lambda \, xs. \lambda \, ys. \\
match \, xs \, with \\
   Nil \rightarrow Nil \\
   Cons \, x \, xt \rightarrow \\
      \text{if} \, !(\text{member} \, x \, ys) \\
         \text{then} \, common \, xt \, ys \\
      \text{else} \, \text{Cons} \, x \, (\text{common} \, xt \, ys) \\
match \, ys \, with \\
   Nil \rightarrow Nil \\
   Cons \, y \, yt \rightarrow \\
      \text{if} \, x < y \\
         \text{then} \, common \, xt \, ys \\
      \text{else if} \, y < x \\
         \text{then} \, common \, xs \, yt \\
      \text{else} \, \text{Cons} \, x \, (\text{common} \, xs \, ys)
What we have

“Find the intersection of two sorted lists”

Synthesizer

$O(m \cdot n)$

What we want

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

Synthesizer

$O(m + n)$
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"
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: AARA

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 \text{v} \geq 0 \}
Refinement types

common = ??
Refinement types

common :: xs: SList a \rightarrow ys: SList a \\
\rightarrow v: \{\text{List } a \mid \text{elems } v = \text{elems } xs \cap \text{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 \rightarrow ys: SList a
          \rightarrow v: \{ \text{List } a \mid \text{elems } v = \text{elems } xs \cap \text{elems } ys\}
common = ??
Synquid

Functional specification

Library functions

[Polikarpova et. al, 2016]
Functional specification

Library functions

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)

[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**”

Refinement: boolean

Potential: numeric
Resource annotations

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

common :: xs: SList \texttt{a}^1 \rightarrow ys: SList \texttt{a}^1
\rightarrow v: \{\text{List a} \mid \text{elems v} = \text{elems xs} \cap \text{elems ys}\}
common = ??
Synthesize with ReSyn

common :: xs: SList a₁ → ys: SList a₁
             → v: {List a | elems v = elems xs ∩ elems ys}
common = ??

member
Cons, Nil, ...
≤, =, !, ...
Components: **member**

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

member :: z:a \to zs:\text{List } a^1
\to v:\{\text{Bool}|v = (x \in \text{elems } xs)\}
Components: member

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

Resource bound

Library functions

ReSyn
ReSyn

common = \lambda xs. \lambda ys.
match xs with
  Nil \rightarrow Nil
  Cons x xt \rightarrow
    match ys with
      Nil \rightarrow Nil
      Cons y yt \rightarrow
        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?

\[
\text{common} = \lambda \text{xs.} \; \lambda \text{ys.} \\
\text{match xs with} \\
\text{Nil } \rightarrow \text{ Nil} \\
\text{Cons x xt } \rightarrow \\
\text{ if } !\text{(member x ys)} \\
\quad \text{then } \text{common xt ys} \\
\text{else } \text{Cons x (common xt ys)}
\]
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 a
        → v:{Bool|v = (x ∈ elems xs)}
How do we automate this reasoning?

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

```plaintext
match \; xs \; with
  Nil \rightarrow \text{Nil}
  Cons \; x \; xt \rightarrow
    \text{if} \; !\text{(member \; x \; ys)}
    \text{then} \; \text{common} \; xt \; ys
    \text{else} \; \text{Cons} \; x \; (\text{common} \; xt \; 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?

\[
\text{common} = \lambda \text{xs. } \lambda \text{ys. } \begin{cases} \text{match } \text{xs with} \\
\text{Nil} \rightarrow \text{Nil} \\
\text{Cons } x \text{ xt} \rightarrow \\
\text{if } !\text{(member } x \text{ ys)} \\
\text{then } \text{common xt ys} \\
\text{else } \text{Cons } x \text{ (common xt ys)} \end{cases}
\]
\[
\text{common} = \lambda \, \text{xs}. \; \lambda \, \text{ys}.
\]
\[
  \textbf{match} \; \text{xs} \; \textbf{with} \\
  \quad \text{Nil} \rightarrow \text{Nil} \\
  \quad \text{Cons} \; x \; \text{xt} \rightarrow \\
    \hspace{1em} \textbf{if} \; !(\text{member} \; x \; \text{ys}) \\
    \hspace{2em} \textbf{then} \; \text{common} \; \text{xt} \; \text{ys} \\
    \hspace{1em} \textbf{else} \; \text{Cons} \; x \; (\text{common} \; \text{xt} \; \text{ys})
\]
common = \lambda\, \textit{xs}. \lambda\, \textit{ys}.

\textbf{match} \, \textit{xs} \, \textbf{with}
  Nil \rightarrow \text{Nil}
  \text{Cons} \, \textit{x} \, \textit{xt} \rightarrow
    \textbf{if} \, \neg\, \text{(member} \, \textit{x} \, \, (\textit{ys} \, :: \, \textbf{List} \, \textit{a}^p)) \, \textbf{then} \, \text{common} \, \textit{xt} \, \, (\textit{ys} \, :: \, \textbf{List} \, \textit{a}^q) \, \textbf{else} \, \text{Cons} \, \textit{x} \, \, \text{(common} \, \textit{xt} \, \, \textit{ys})
common = \lambda \text{xs.} \lambda \text{ys.}

\textbf{match} \ \text{xs with}
\text{Nil} \rightarrow \text{Nil}
\text{Cons} \ x \ \text{xt} \rightarrow
\begin{align*}
\quad & \text{if } \neg (\text{member } x (\text{ys :: List } a)) \\
\quad & \quad \text{then } \text{common } xt \ \text{ys} \\
\quad & \quad \text{else } \text{Cons} \ x \ (\text{common } xt \ \text{ys})
\end{align*}
member :: z:a → zs: List a^1 → 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)
member :: z:a \to \textbf{zs: List a}^1 \to v:\{\text{Bool}|\ldots\}

\begin{align*}
\text{common} &= \lambda \text{xs. } \lambda \text{ys.} \\
\text{match} & \text{ xs with} \\
\text{Nil } \to \text{ Nil} \\
\text{Cons } \text{x } \text{ xt } \to \\
\text{ if } !\text{(member x (ys :: List a^p))} \\
\text{ then } \text{ common } \text{ xt } \text{ ys} \\
\text{ else } \text{ Cons } \text{x } \text{(common } \text{ xt } \text{ ys)}
\end{align*}

\text{List a}^p \text{ } <:: \text{ List a}^1
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)
\[ a \leq b \quad p \geq q \]

\[
\text{List } a^p \leq: \text{List } b^q
\]
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 a⁹)
        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⁺)
      else Cons x (common xt ys)
Sharing $\rightarrow$ \texttt{SList a}^1 $\Downarrow$ \texttt{SList a}^p, \texttt{SList a}^q

\[
\text{common} = \lambda \text{xs. } \lambda \text{ys.} \\
\text{match xs with} \\
\text{Nil } \rightarrow \text{ Nil} \\
\text{Cons x xt } \rightarrow \\
\text{if } !(\text{member x ys}) \text{ then common xt ys} \\
\text{else Cons x (common xt ys)}
\]
Sharing \rightarrow \text{SList } a^1 \Downarrow \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{ } \text{xt} \rightarrow
\quad \text{if } ! (\text{member } x \text{ } \text{ys})
\quad \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. \\
\text{match} \, xs \, \text{with} \\
\text{Nil} \rightarrow \text{Nil} \\
\text{Cons} \, x \, xt \rightarrow \\
\quad \text{if} \, ! (\text{member} \, x \, ys) \\
\qquad \text{then} \, \text{common} \, xt \, ys \\
\qquad \text{else} \, \text{Cons} \, x \, (\text{common} \, xt \, ys)
SMT

\[ 1 = p + q \land \]
\[ p \geq 1 \land \]
\[ q \geq 1 \]
SMT

\[ 1 = p + q \land p \geq 1 \land q \geq 1 \]
This talk

1. Specification
2. Analysis
3. Search
Synthesis is search
Synthesis is search
Synthesis is search
Synthesis is search
Reject impossible programs early

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

\text{match} \ \text{xs} \ \text{with}
\begin{align*}
\text{Nil} & \rightarrow \text{Nil} \\
\text{Cons} \ x \ \text{xt} & \rightarrow \begin{cases} \\
\text{if} \ !((\text{member} \ x \ \text{ys})) & \text{then} \ \text{common} \ \text{xt} \ \text{ys} \\
\text{else} & ?? \end{cases}
\end{align*}
A different approach
Reject impossible programs early with local analysis

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

\[
\text{match } \mathbf{xs} \text{ with }
\begin{align*}
\text{Nil} & \rightarrow \text{Nil} \\
\text{Cons } x \ x\mathbf{t} & \rightarrow \\
\text{if } !(\text{member } x \ \mathbf{ys}) & \\
\text{then } \text{common } x\ \mathbf{t} \ \mathbf{ys} & \\
\text{else } ?? &
\end{align*}
\]
Reject impossible programs early with local analysis

common = λ xs. λ ys.
match xs with
  Nil → Nil
  Cons x xs →
    if !(member x ys)
    then common xs ys
    else ??
Reject impossible programs early with local analysis

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

1. Can ReSyn generate faster programs than Synquid?
Results

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

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

```plaintext
compress xs =
    match xs with
    Nil -> Nil
    Cons x3 x4 ->
        match compress x4 with
        Nil -> Cons x3 Nil
        Cons x10 x11 ->
            if x3 == x10
                then compress x4
                else Cons x3 (Cons x10 x11)
```

**O(2^n)**

Synquid

**O(n)**

ReSyn
**insert**: Insert into a sorted list

\[
\text{insert } x \: x s = \begin{cases} 
\text{match } x s \text{ with} \\
\text{Nil} \rightarrow \text{Cons } x \: \text{Nil} \\
\text{Cons } y \: y s \rightarrow \\
\text{if } x < y \\
\text{then } \text{Cons } x \: (\text{insert } y \: y s) \\
\text{else } \text{Cons } y \: (\text{insert } x \: y s)
\end{cases}
\]

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

**Synquid**

\[
\text{insert } x \: x s = \begin{cases} 
\text{match } x s \text{ with} \\
\text{Nil} \rightarrow \text{Cons } x \: \text{Nil} \\
\text{Cons } y \: y s \rightarrow \\
\text{if } x < y \\
\text{then } \text{Cons } x \: (\text{Cons } y \: y s) \\
\text{else } \text{Cons } y \: (\text{insert } x \: y s)
\end{cases}
\]

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

**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)

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

O(n) → O(n)

“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?
3. What happens if the analysis is non-local?

- ReSyn
- Enumerate-and-check

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

Timeout > 600s

common = \lambda xs. \lambda ys. 
\begin{cases} 
\text{match } xs \text{ with} \\
\text{Nil } \rightarrow \text{ Nil} \\
\text{Cons } x \ xt \rightarrow \\
\text{if } !(\text{member } x \ ys) \\
\text{then } \text{common } xt \ ys \\
\text{else } \text{Cons } x \ (\text{common } x \ ys) 
\end{cases}
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