Resource-Guided Program Synthesis

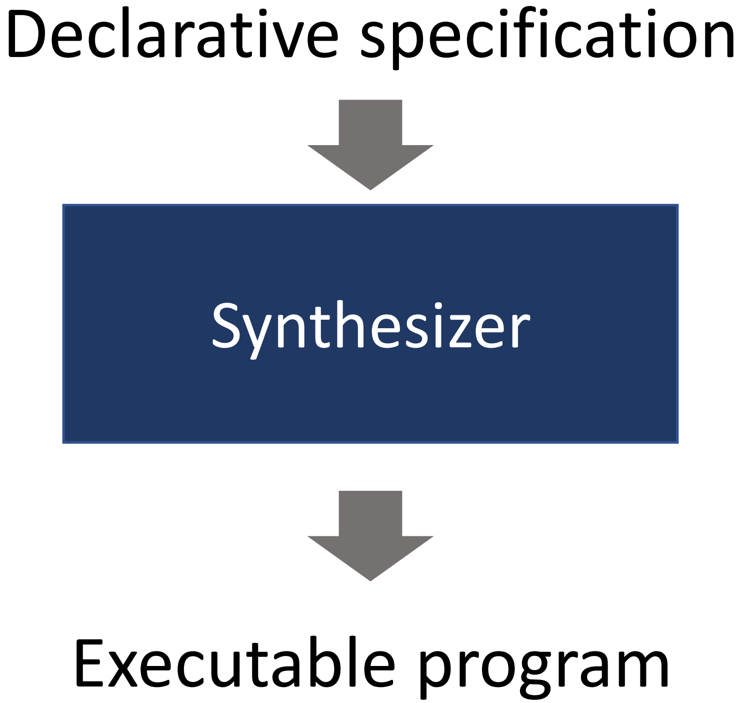
Tristan Knoth[1], Di Wang[2], Nadia Polikarpova[1], Jan Hoffmann[2]

[1]UC San Diego

[2]Carnegie Mellon University

PLDI 2019

Program Synthesis



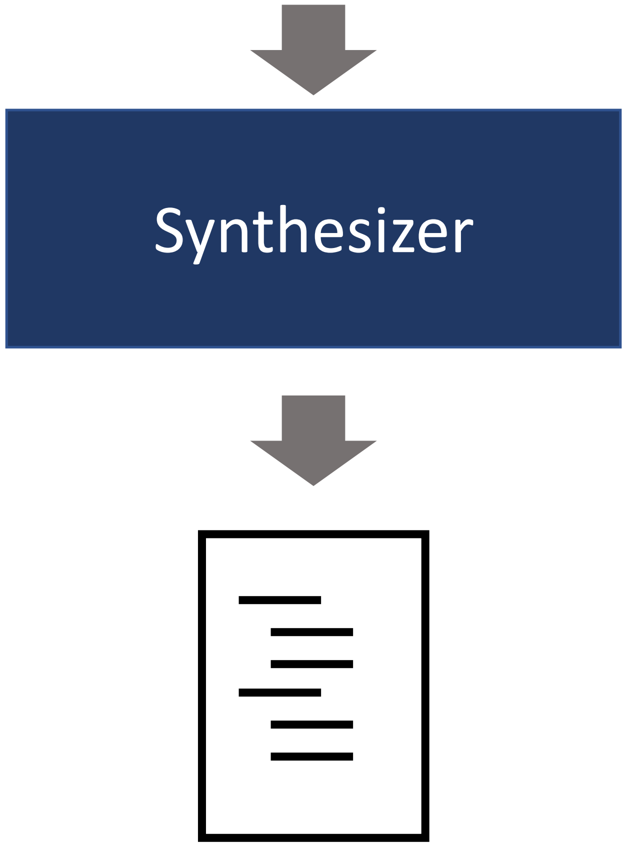
Declarative specification

Synthesizer

Executable program

State of the art

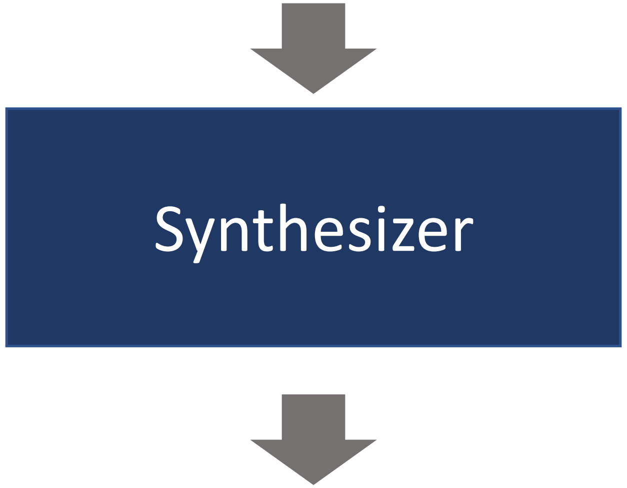
“Find the intersection of two **sorted** lists”



Synthesizer

Type-directed synthesis

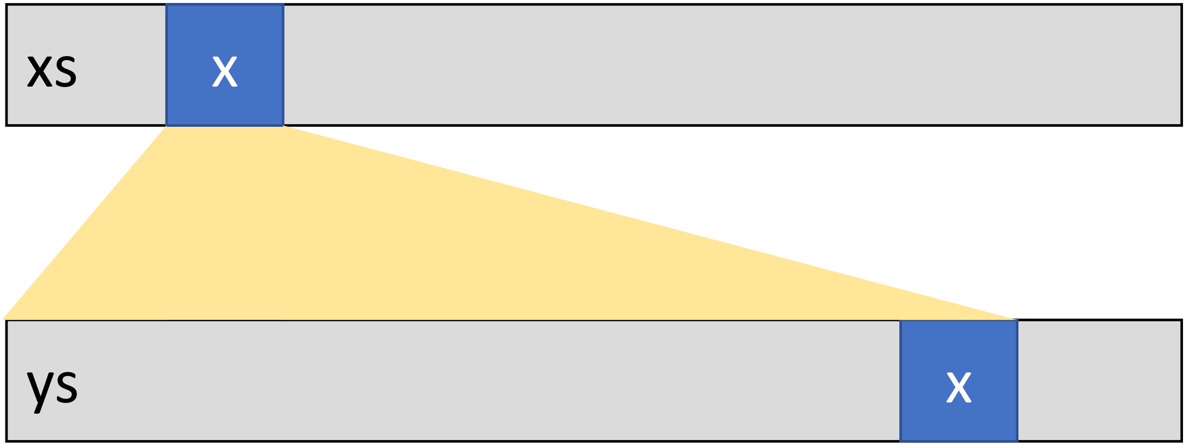
```
common :: xs: SList a → ys: SList a
       → v: {List a | elems v = elems xs ∩ elems ys}
```



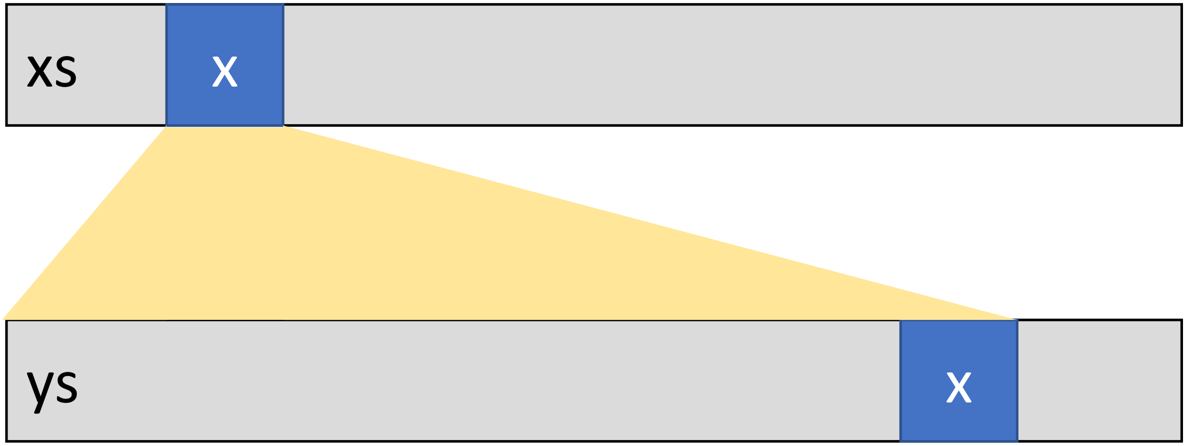
Synthesizer

```
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)
```

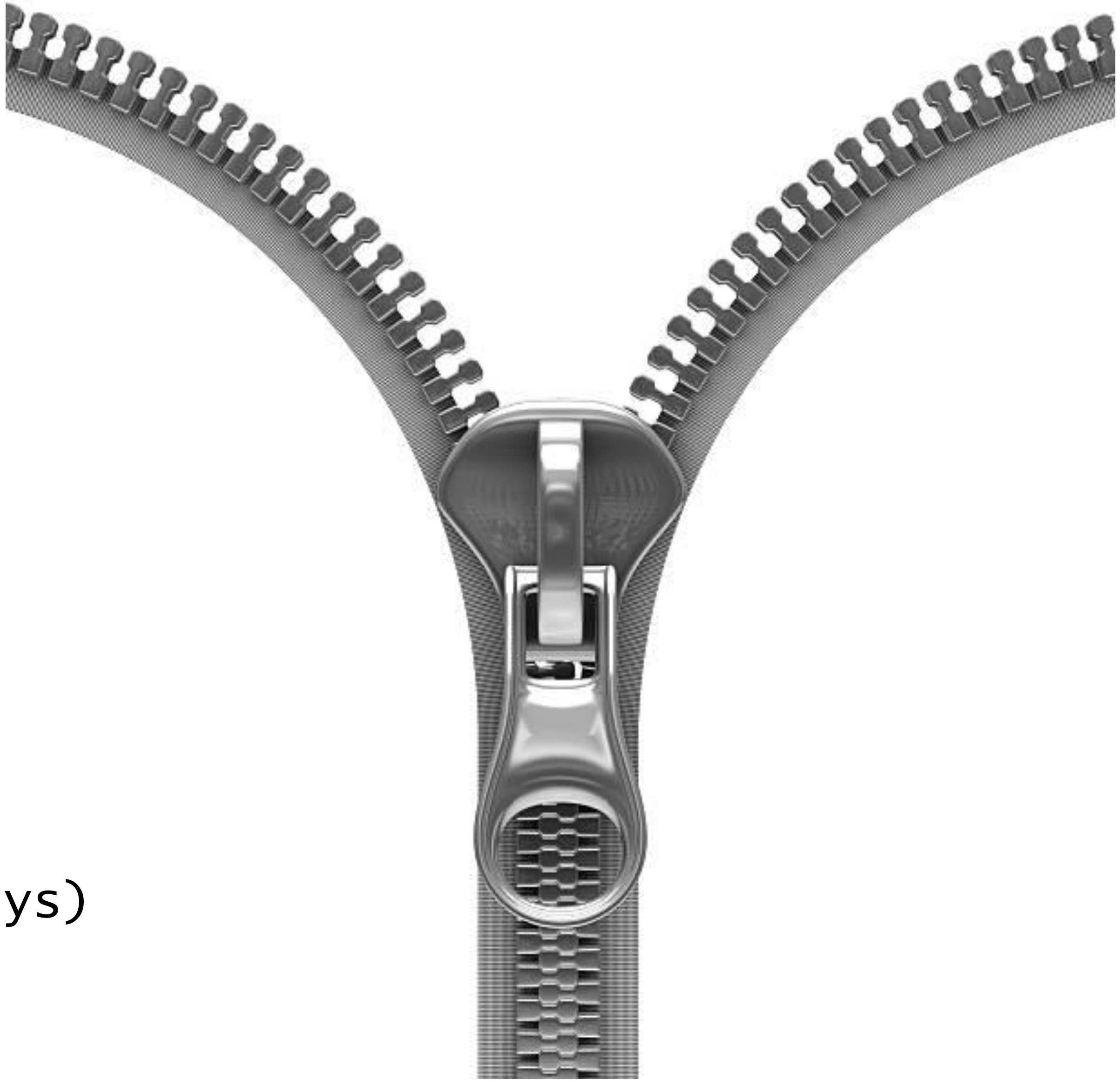
```
common = λ xs. λ ys.
  match xs with
    Nil → Nil
    Cons x xt →
      if !(member x ys)
        then common xt ys
        else Cons x (common xt ys)
```



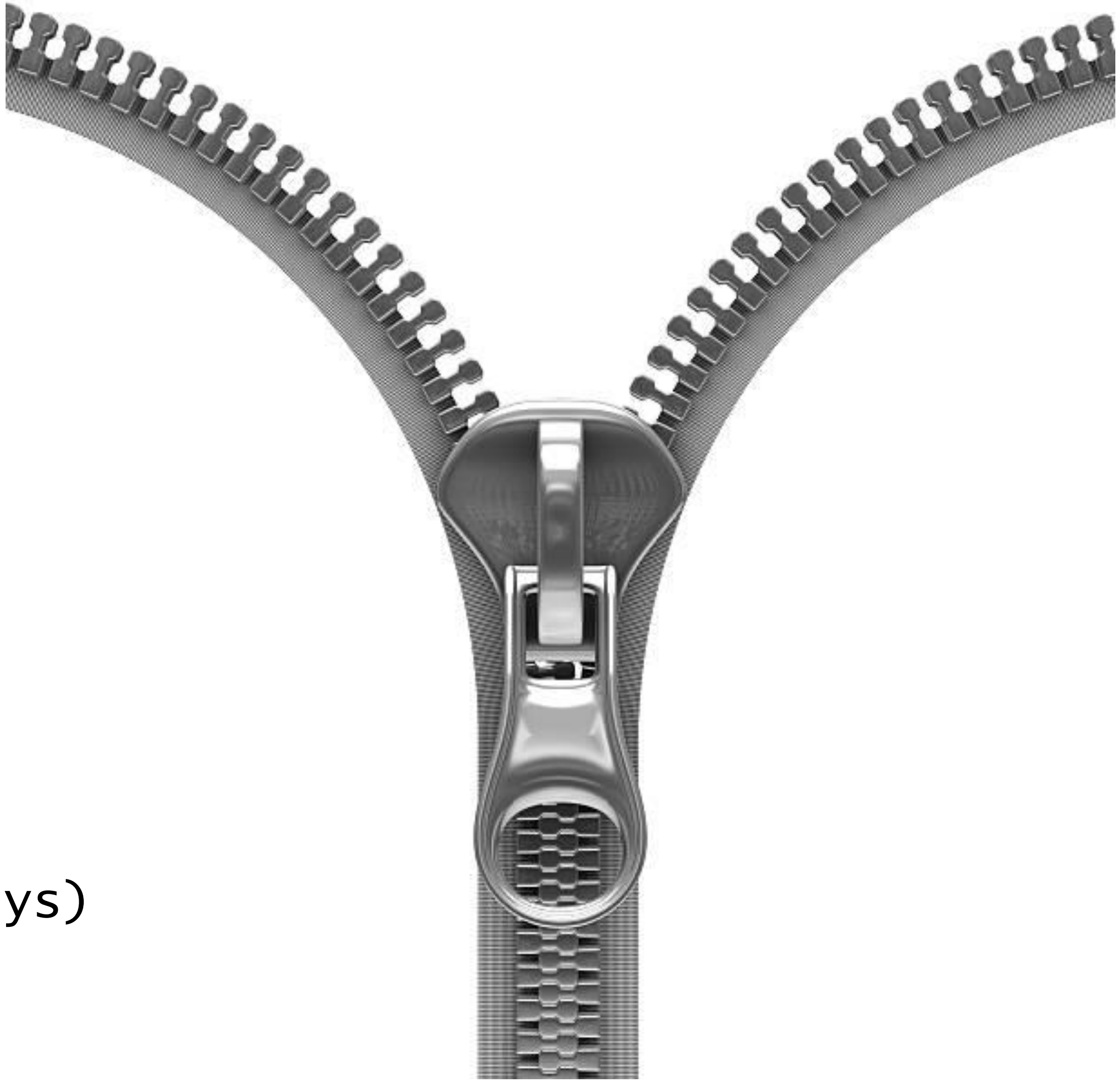
```
common = λ xs. λ ys.
  match xs with
    Nil → Nil
    Cons x xt →
      if !(member x ys)
        then common xt ys
        else Cons x (common xt ys)
```



```
common = λ xs. λ ys.
  match xs with
    Nil → Nil
    Cons x xt →
      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)
```



```
common = λ xs. λ 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)

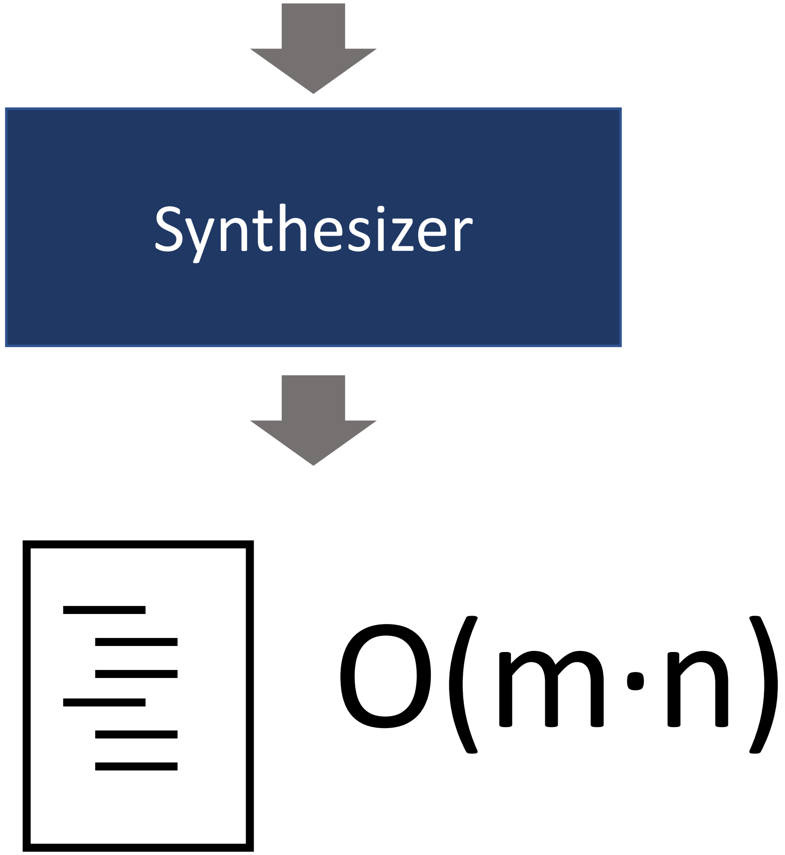
```
common = λ xs. λ ys.
  match xs with
    Nil → Nil
    Cons x xt →
      if !(member x ys)
        then common xt ys
        else Cons x (common xt ys)
```

O(m + n)

```
common = λ xs. λ 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)
```

What we have

“Find the intersection of two sorted lists”

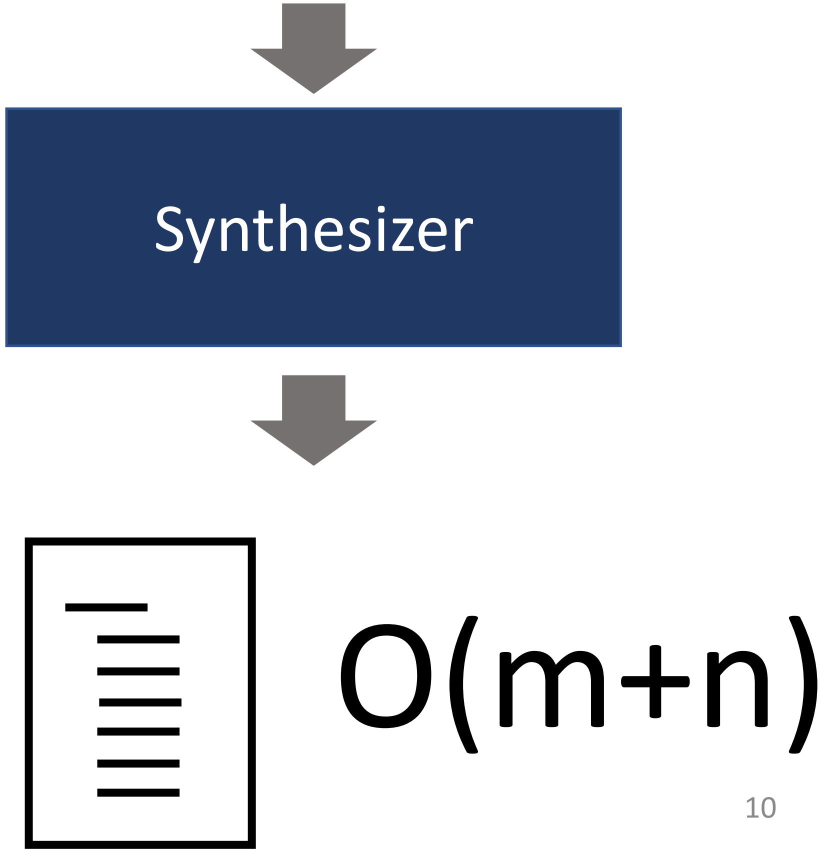


Synthesizer

O(m·n)

What we want

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



Synthesizer

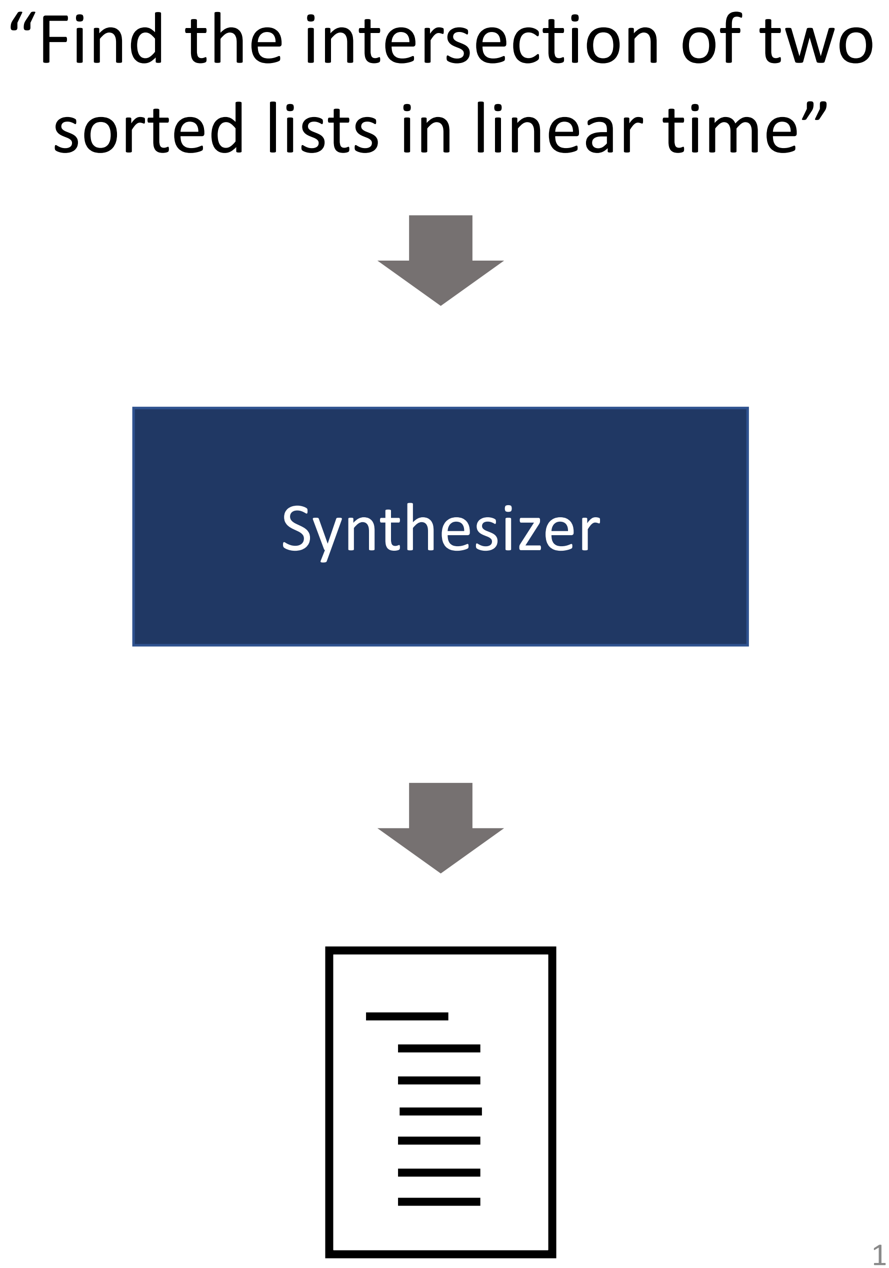
O(m+n)

ReSyn

The first resource-aware synthesizer for recursive programs

This talk

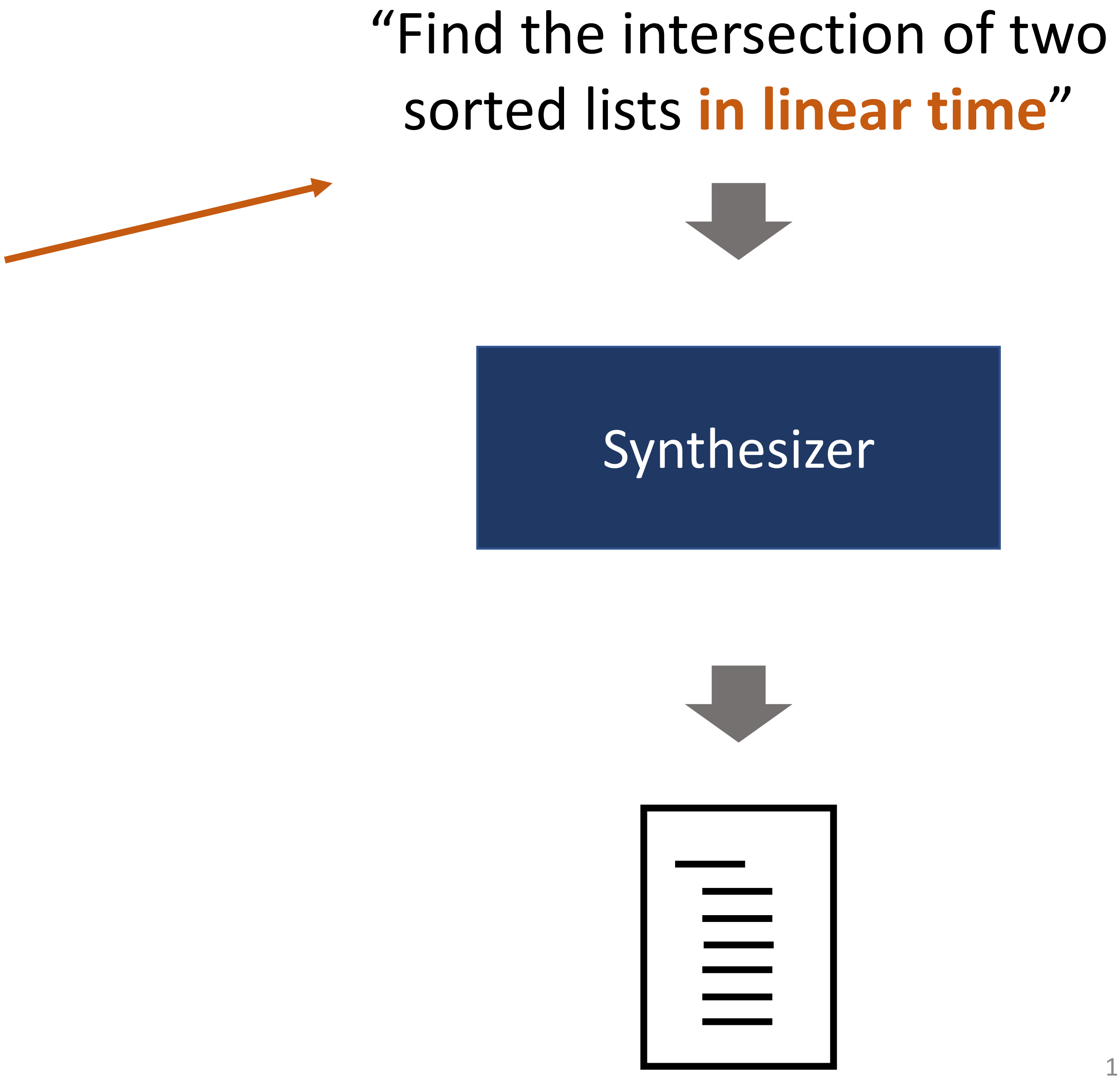
1. Specification



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

This talk

1. Specification



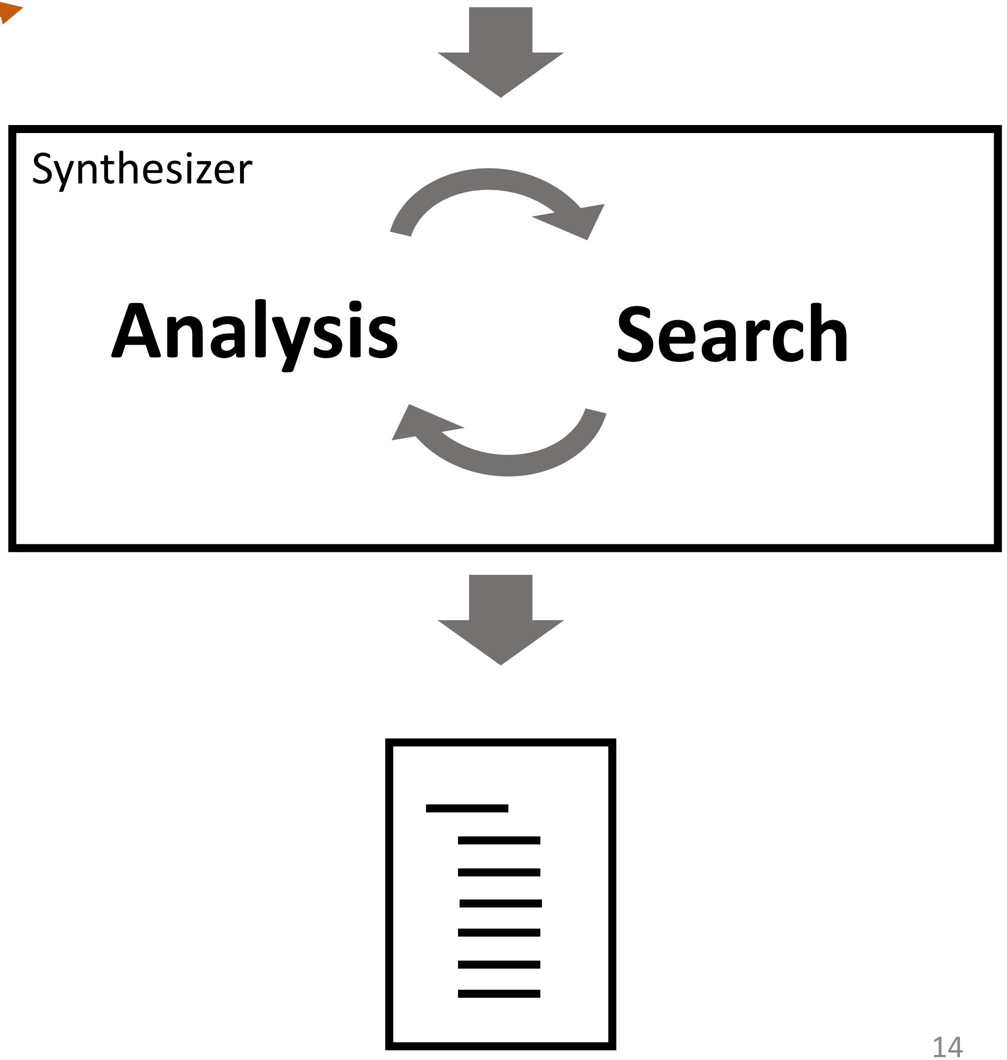
“Find the intersection of two sorted lists **in linear time**”

Synthesizer

This talk

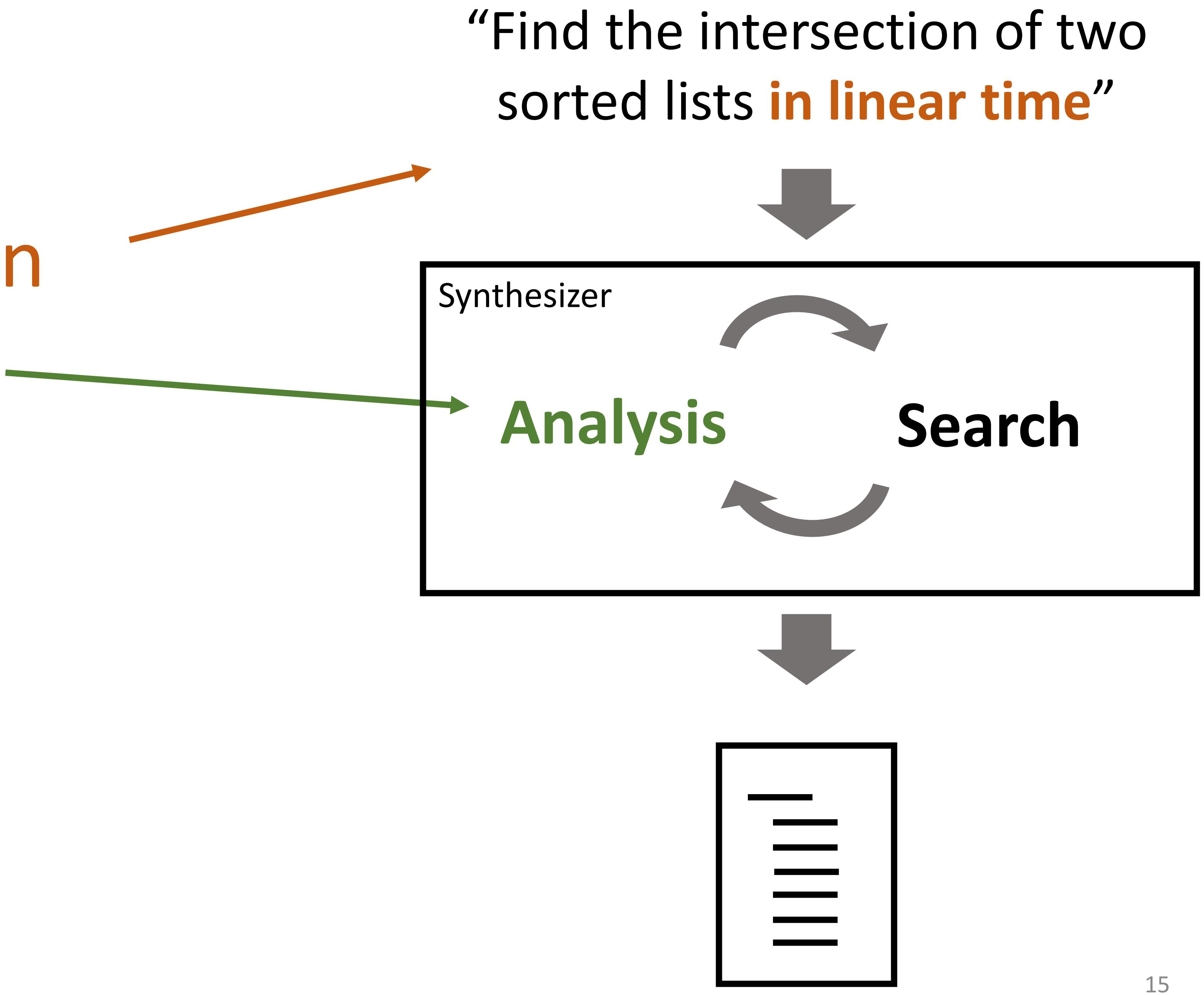
1. Specification

“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**”

Synthesizer

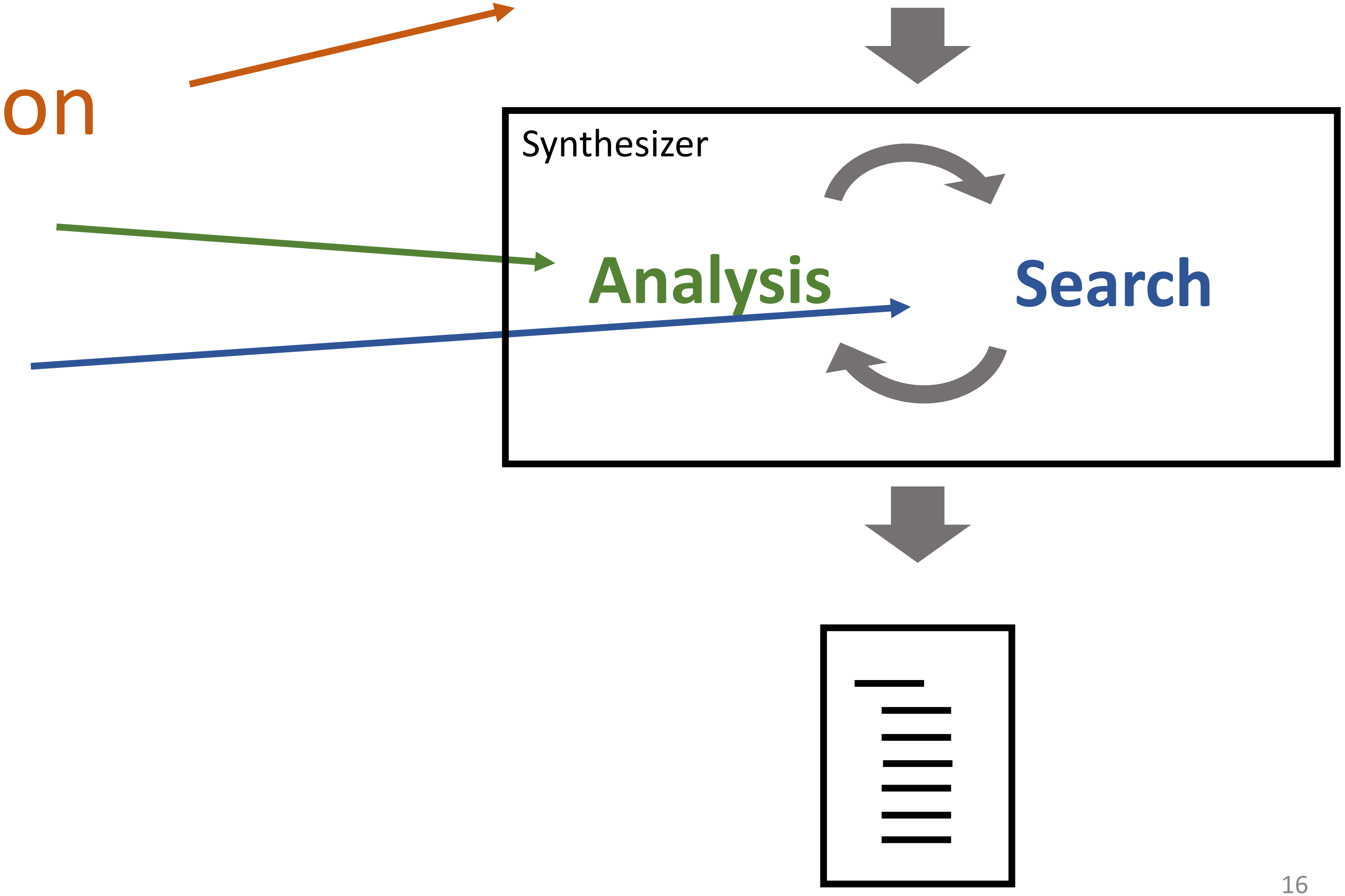
Analysis

Search

This talk

1. Specification
2. Analysis
3. Search

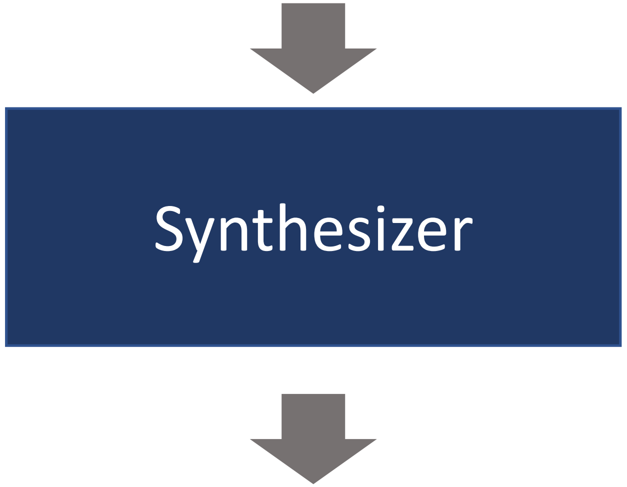
"Find the intersection of two sorted lists **in linear time**"



This talk

1. Specification
2. Analysis
3. Search

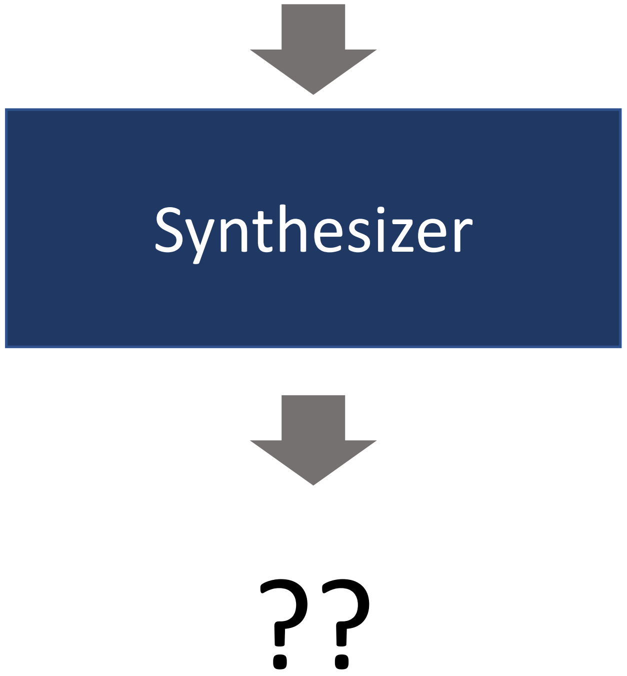
“Find the intersection of two sorted lists in linear time”



Synthesizer

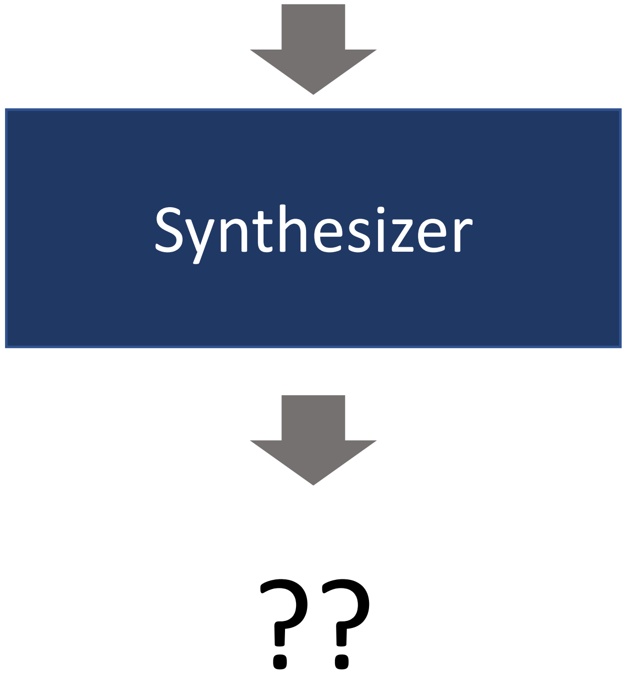
??

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



Refinement types

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

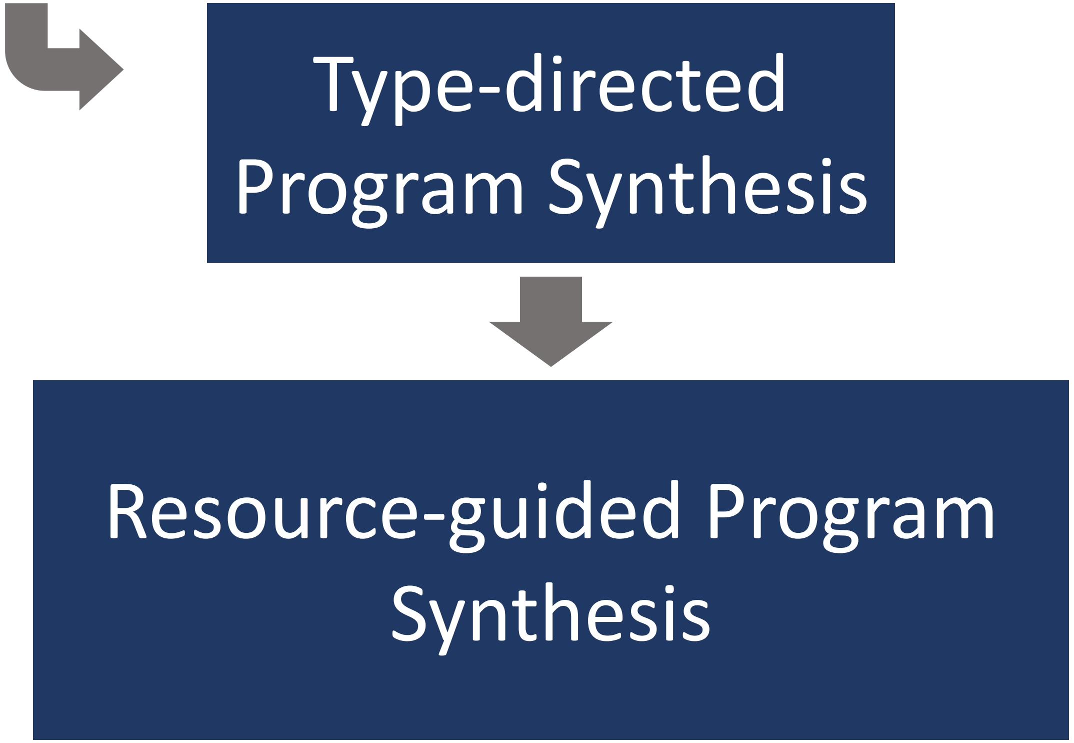

Synthesizer

??

Refinement types
with
Resource annotations

Refinements:

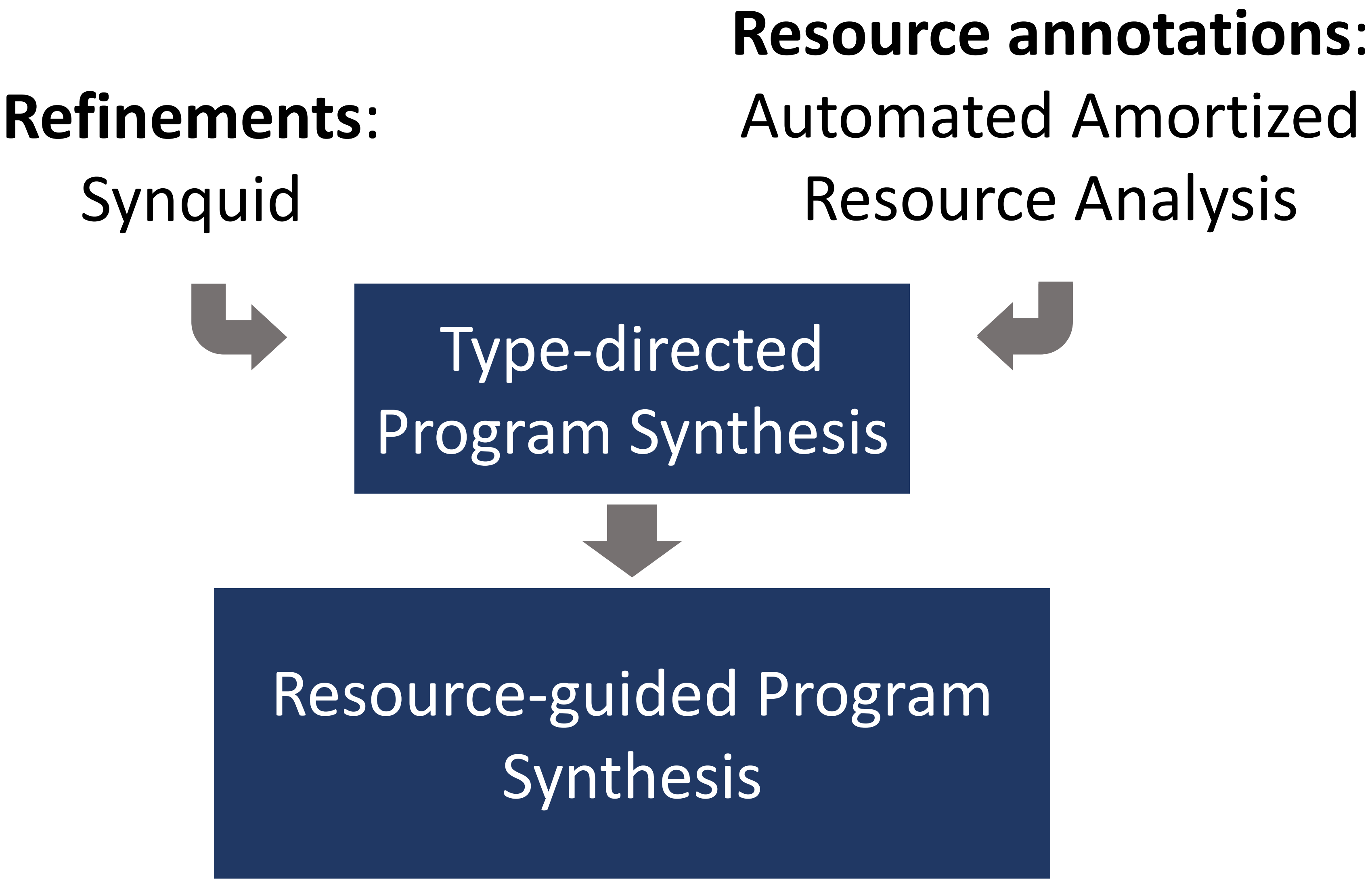
Synquid



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”

$$\{B \mid \Psi\}$$

```
v:{Int|v≥0}
```

Refinement types

```
common = ??
```

Refinement types

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

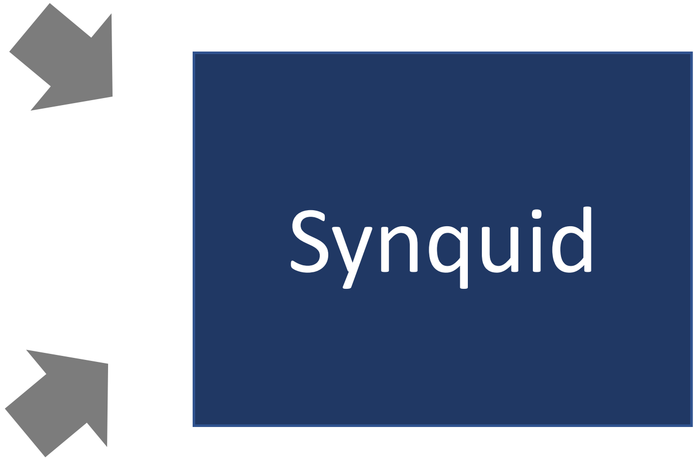
Refinement types

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

Refinement types

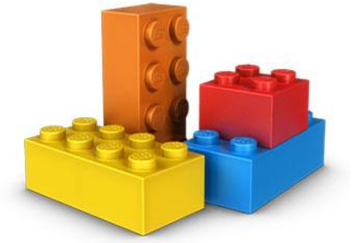
```
common :: xs: SList a → ys: SList a
       → v: {List a | elems v = elems xs ∩ elems ys}
common = ??
```

Functional specification



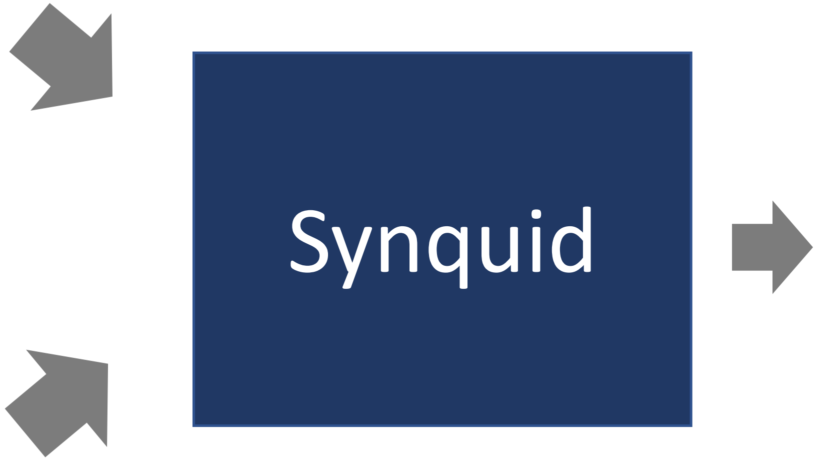
Synquid

Library functions



[Polikarpova et. al, 2016]

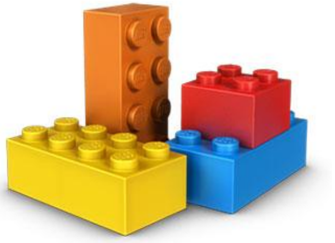
Functional specification



Synquid

```
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)
```

Library functions



[Polikarpova et. al, 2016]

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

$$\{B \mid \Psi\}$$

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

$$\left\{ B \mid \Psi \right\}^{\overset{\substack{\text{Potential}\\ \downarrow}}{\Phi}}$$

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

Potential: numeric

$$\{B \mid \Psi\}^{\Phi}$$

Refinement: boolean

Resource annotations

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

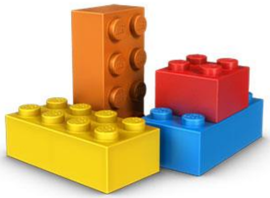
Resource budget

```
common :: xs: SList a^1 → ys: SList a^1
        → v: {List a | elems v = elems xs ∩ 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`

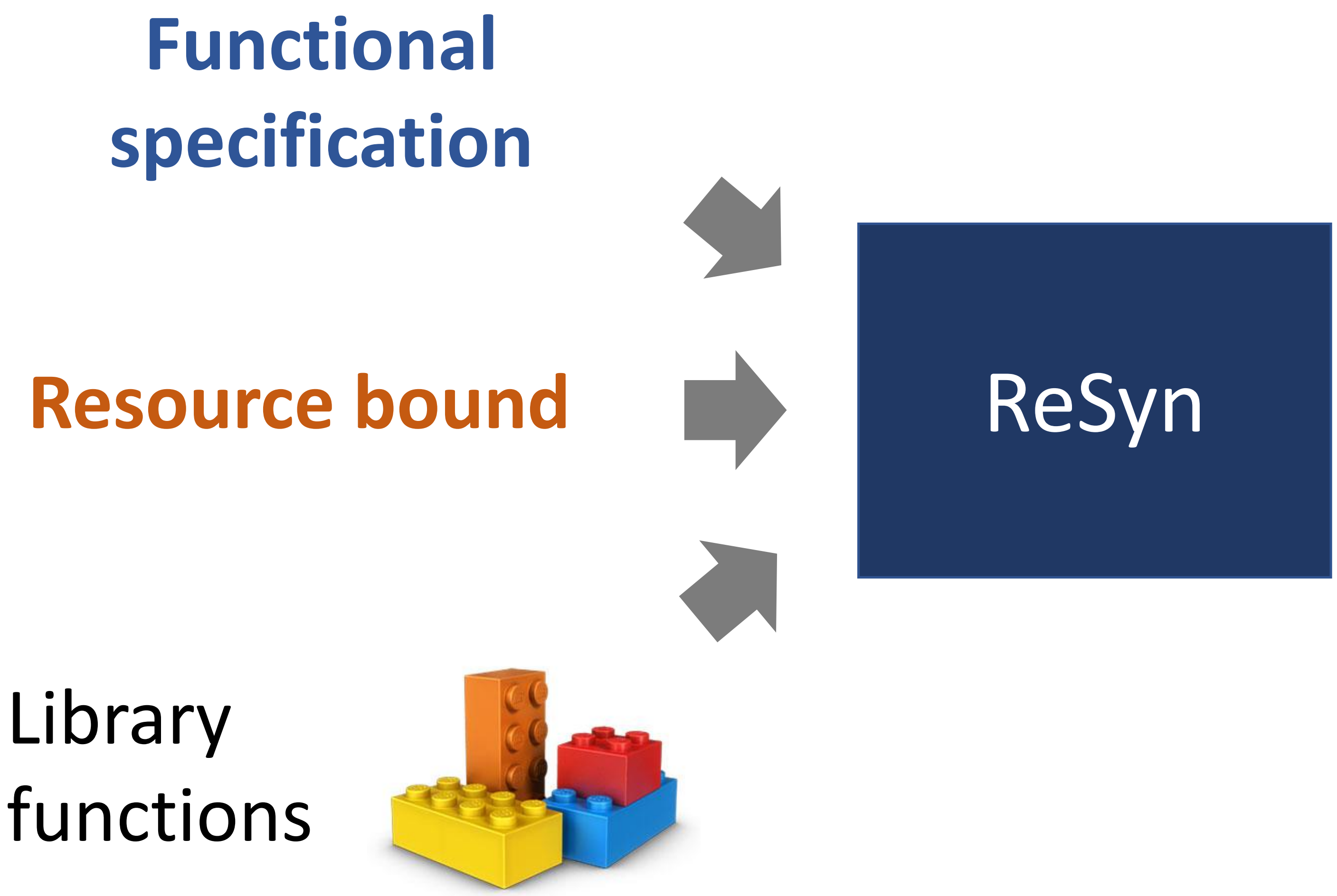
```
member :: z:a → zs: List a
            → ν:{Bool|ν = (x ∈ elems xs)}
```

Components: **member**

```
member :: z:a → zs: List a[1]
         → v:{Bool|v = (x ∈ elems xs)}
```

Components: member

```
member :: z:a → zs: List a[1]
        → ν:{Bool|ν = (x ∈ elems xs)}
```

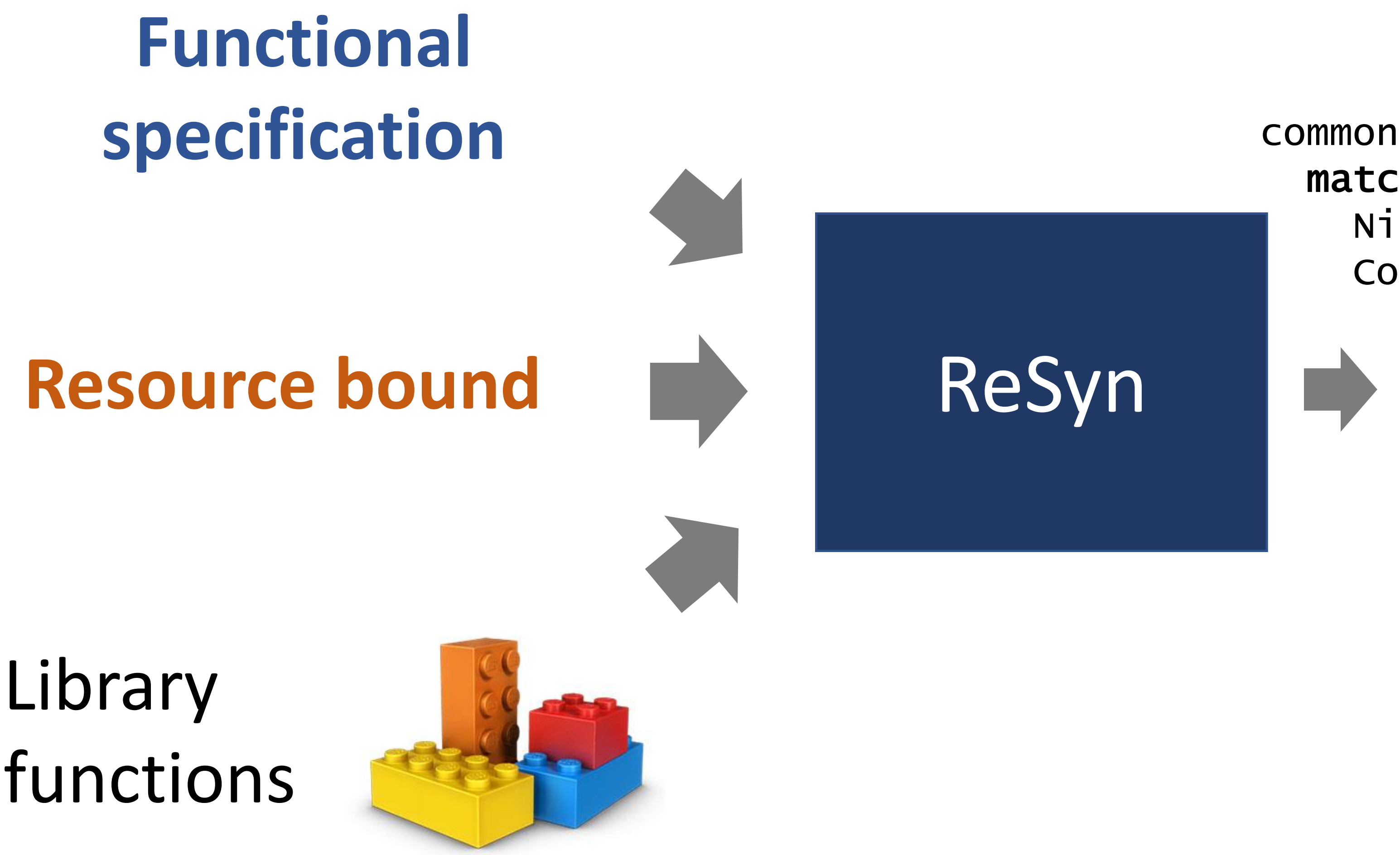


Functional specification

Resource bound

ReSyn

Library functions



Functional specification

Resource bound

Library functions

ReSyn

```
common = λ xs. λ 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)
```

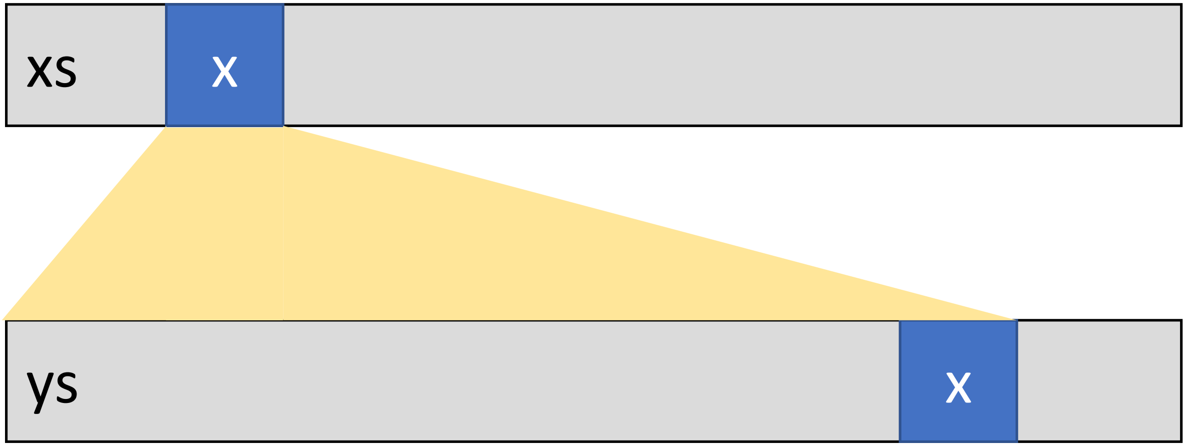
This talk

1. Specification
2. Analysis
3. Search

How do we know common does not run in linear time?

```
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 x ys)
```



```
member :: z:a → zs: List a¹
        → v:{Bool|v = (x ∈ elems xs)}
```

How do we automate this reasoning?

```
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 :: xs: SList a^1 → ys: SList a^1 → ν: {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?

```
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)
```

$\texttt{ys :: SList a}^{1}$

```
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 :: List a^p))
        then common xt (ys :: List a^q)
        else Cons x (common xt ys)
```

```
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 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 → 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)
```

```
List a^p <: List a^1
```

$$\cfrac{\cfrac{\texttt{a <: b} \qquad \texttt{p} \geq \texttt{q}}{\texttt{a}^{\texttt{p}}\ \texttt{<:}\ \texttt{b}^{\texttt{q}}}}{\texttt{List a}^{\texttt{p}}\ \texttt{<:}\ \texttt{List b}^{\texttt{q}}}$$

```
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)
```

$$\text{List } a^{p} <: \text{List } a^{1}$$

$$p \geq 1$$

```
common :: xs: SList $a^1$ → ys: SList $a^1$ → 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)
```

```
common :: xs: SList a^1 → ys: SList a^1 → ν: {List a |…}
```

$$\text{List } a^q <: \text{List } a^1$$
$$q \geq 1$$

```
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)
```

Sharing ⟶ $\texttt{SList a}^{1} \curlyvee \texttt{SList a}^{p}, \texttt{SList a}^{q}$

```
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)
```

Sharing ⟶ $\mathtt{SList\ a}^{1} \curlyvee \mathtt{SList\ a}^{p}, \mathtt{SList\ a}^{q}$

$$1 = p + q$$

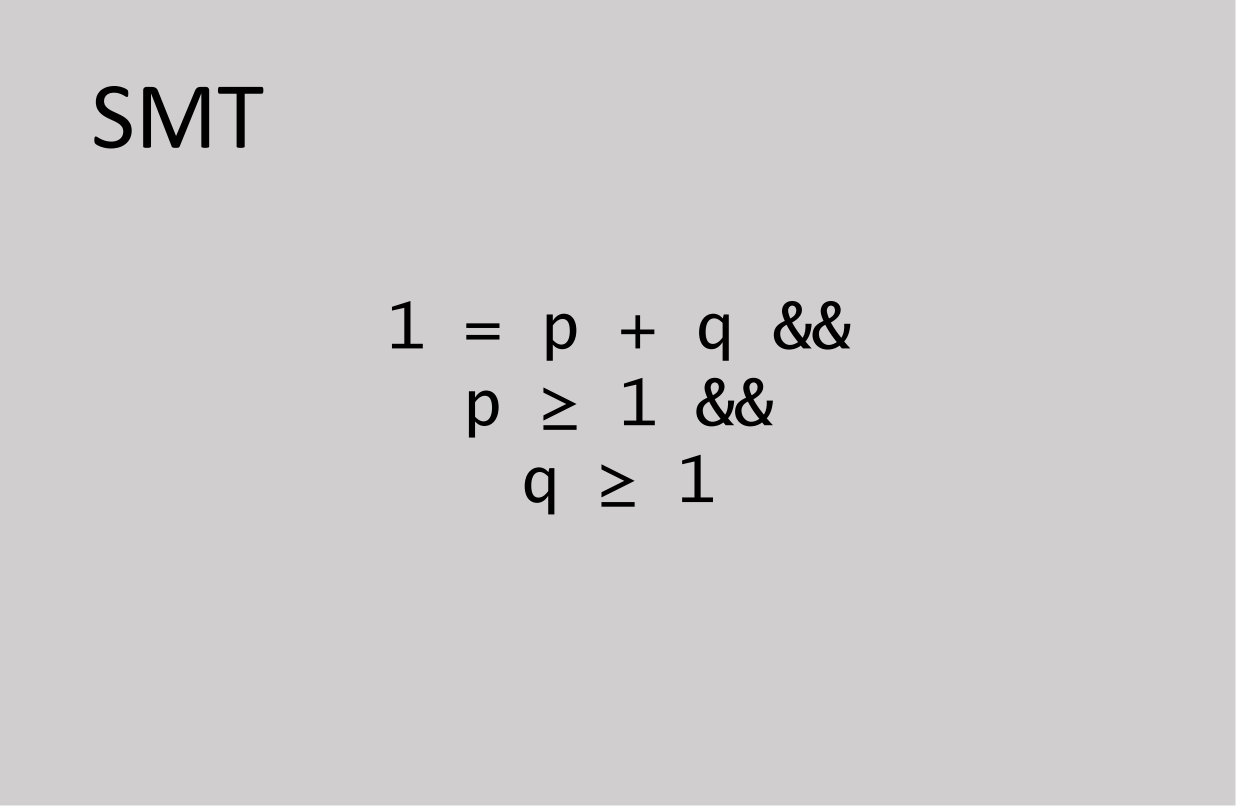
```
common = λ xs. λ ys.
  match xs with
    Nil → Nil
    Cons x xt →
      if !(member x ys)
        then common xt ys
        else Cons x (common xt ys)
```

Sharing → $\mathtt{SList}\ a^{1} \curlyvee \mathtt{SList}\ a^{p},\ \mathtt{SList}\ a^{q}$

$$1 = p + q$$
$$p \geq 1$$
$$q \geq 1$$

Subtyping

```
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)
```



SMT

```
1 = p + q &&
  p ≥ 1 &&
   q ≥ 1
```

SMT

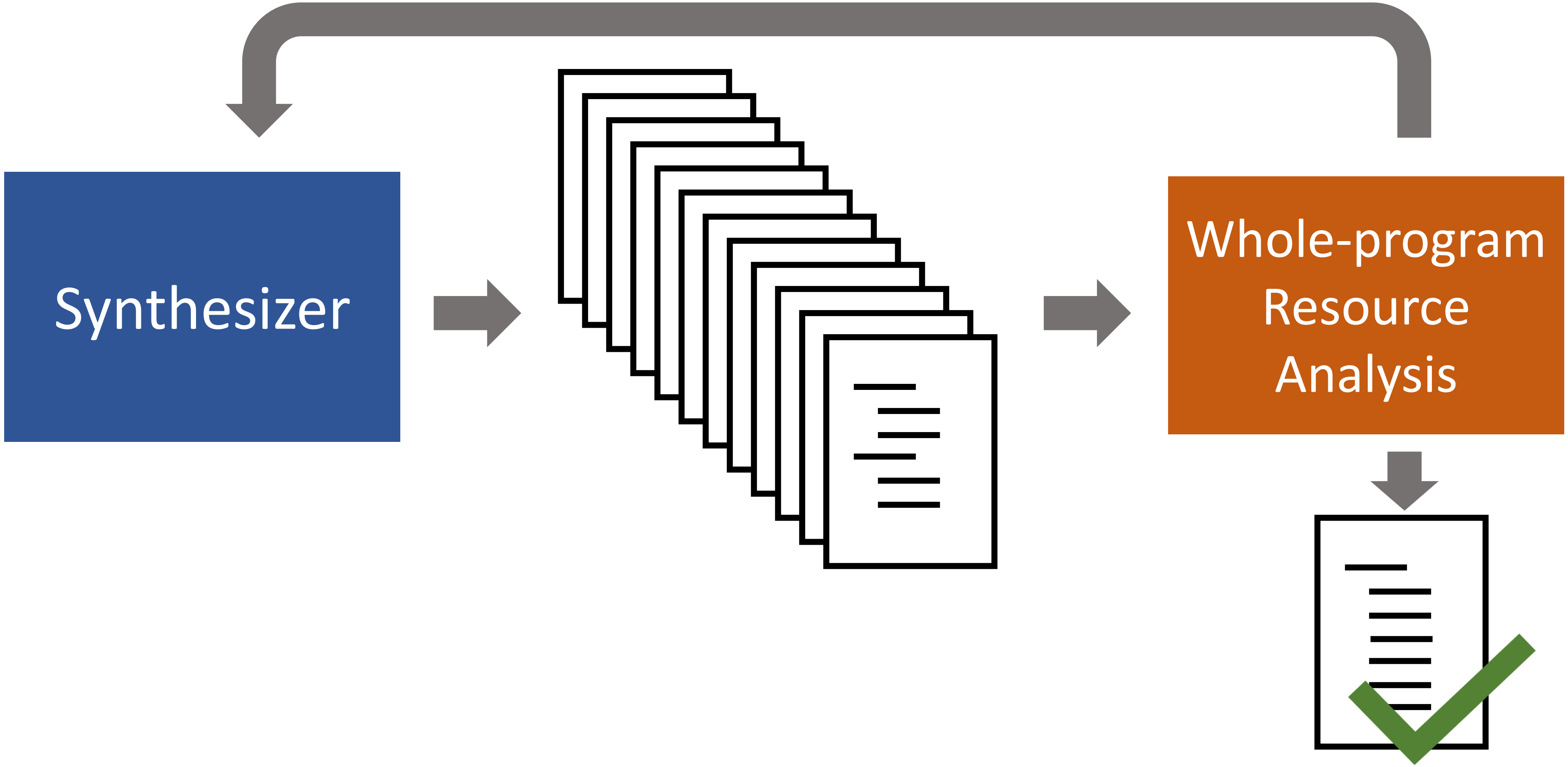
```
1 = p + q &&
  p ≥ 1 &&
    q ≥ 1
```

✗

This talk

1. Specification
2. Analysis
3. Search

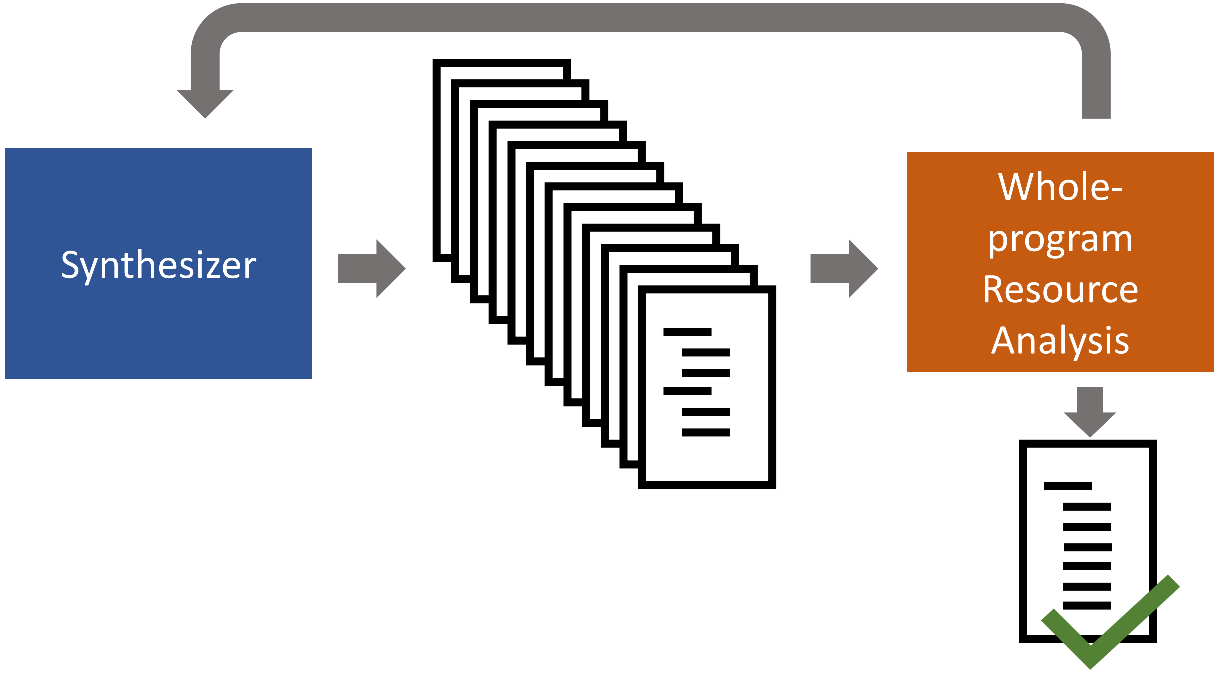
Enumerate-and-check



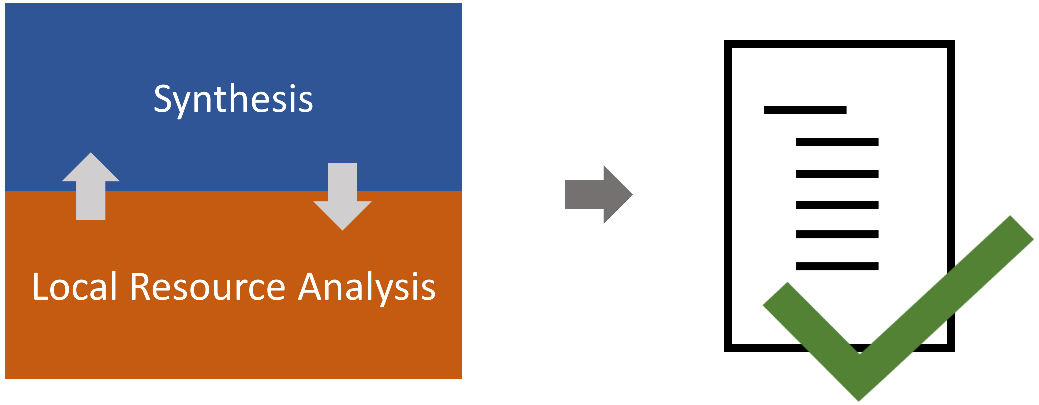
Synthesizer

Whole-program Resource Analysis

Enumerate-and-check



Resource-Guided Synthesis



Reject impossible programs early

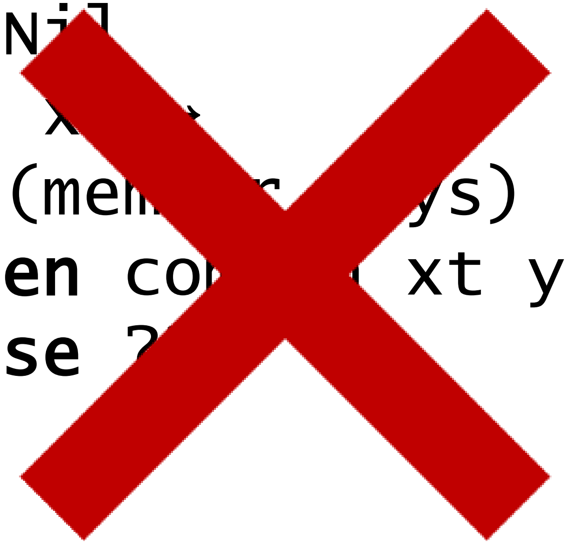
```
common = λ xs. λ ys.
  match xs with
    Nil → Nil
    Cons x xt →
      if !(member x ys)
        then common xt ys
        else ??
```

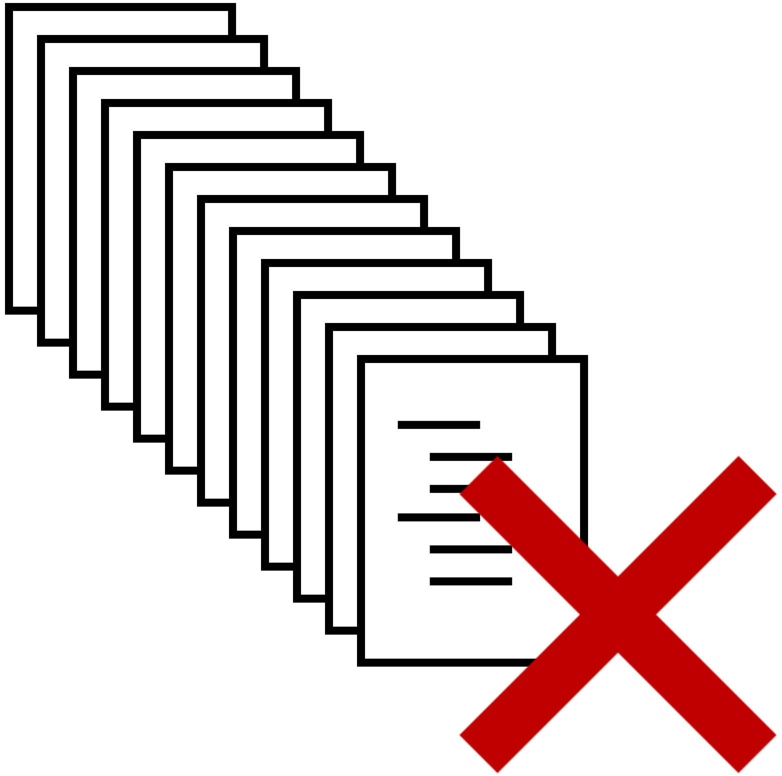
Reject impossible programs early with local analysis

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

Reject impossible programs early with local analysis

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





Reject impossible programs early with local analysis

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

Evaluation

Evaluation

1. Can ReSyn generate faster programs than Synquid?

Evaluation

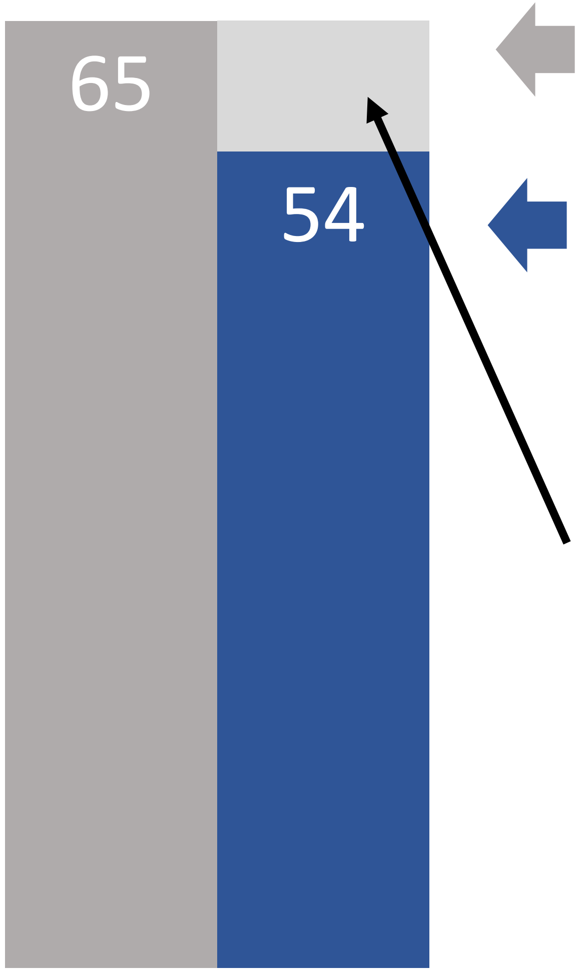
1. Can ReSyn generate faster programs than Synquid?
2. How much longer does ReSyn take to generate code?

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?

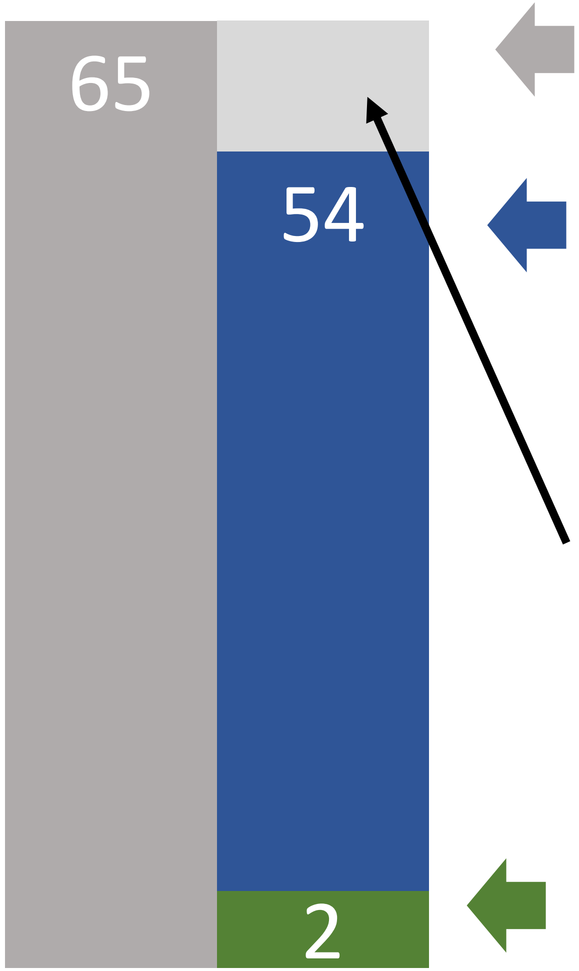


Generated by Synquid

Generated by ReSyn

Require super-linear bound

1. Can ReSyn generate faster programs?



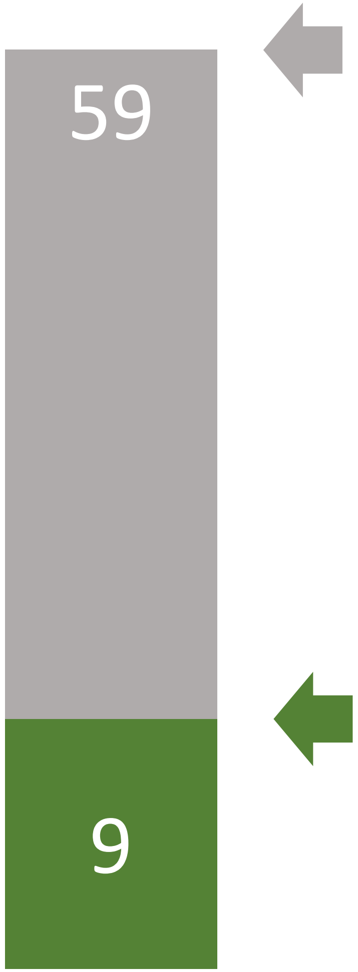
Generated by Synquid

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

```
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)
```

```
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 Cons x10 x11
            else Cons x3 (Cons x10 x11)
```

$O(2^n)$

$O(n)$

Synquid

ReSyn

`insert`: Insert into a sorted list

```
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)

Synquid → ReSyn

$$\texttt{insert :: x:a → xs: SList a}^{\texttt{if x > v then 1 else 0}}$$
$$\texttt{→ v:\{SList a | elems v = elems xs} \cup \texttt{\{x\}\}}$$

```
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)
```

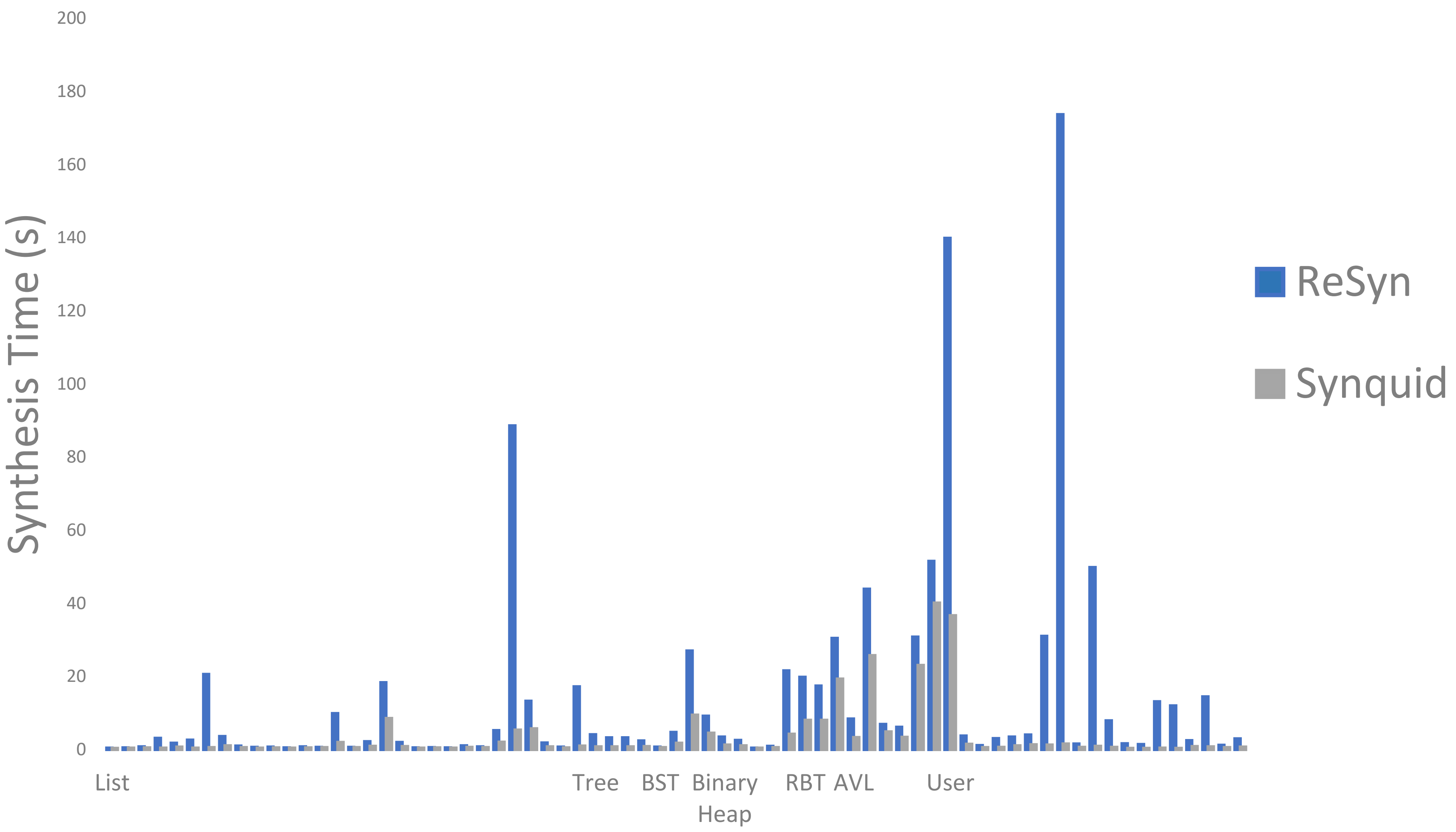
```
insert x xs =
  match xs with
    Nil → Cons x Nil
    Cons y ys →
      if x < y
        then Cons x (Cons y ys)
        else Cons y (insert x ys)
```

O(n)

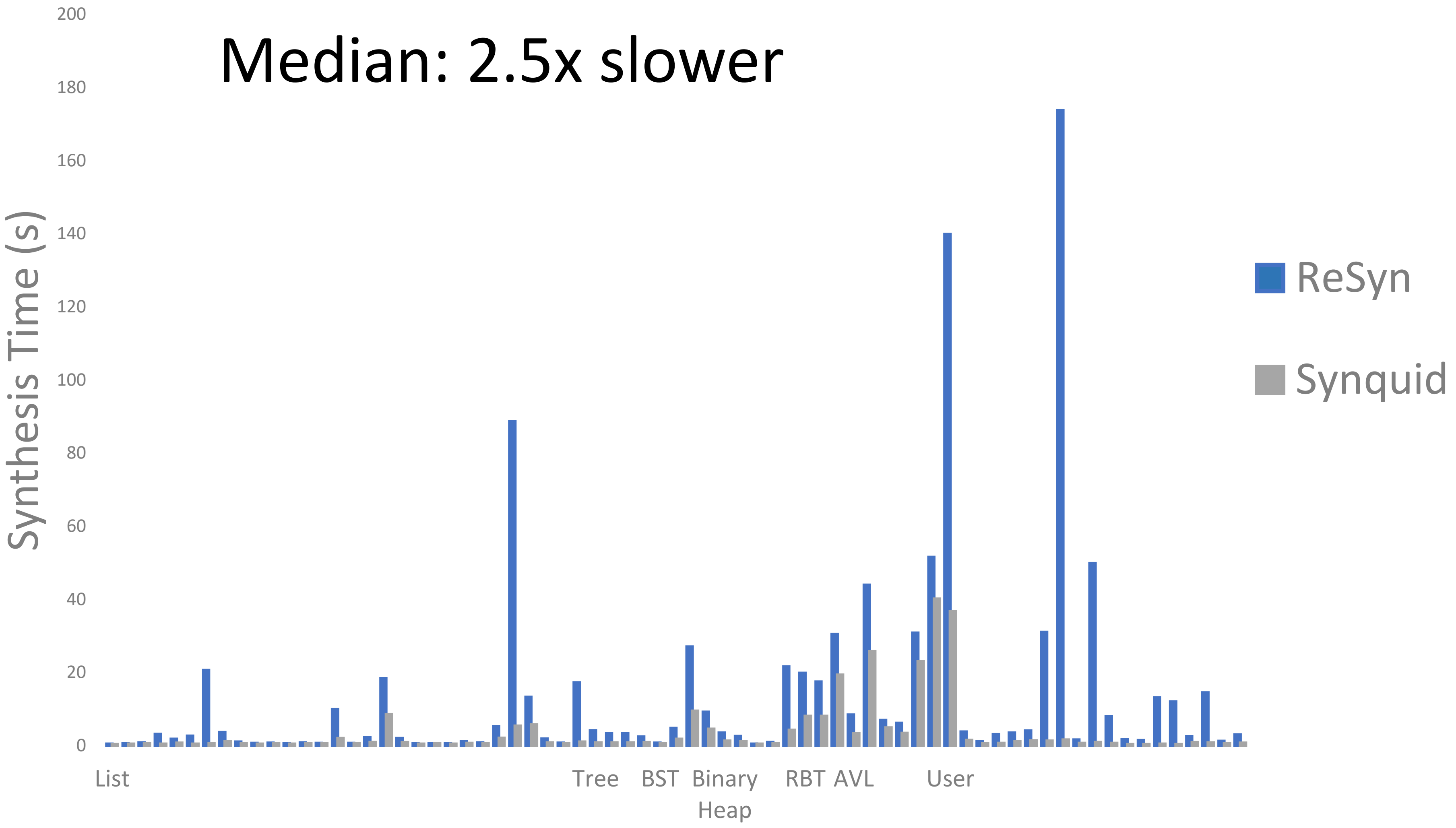
O(n)

“One recursive call per element in xs that is smaller than x”

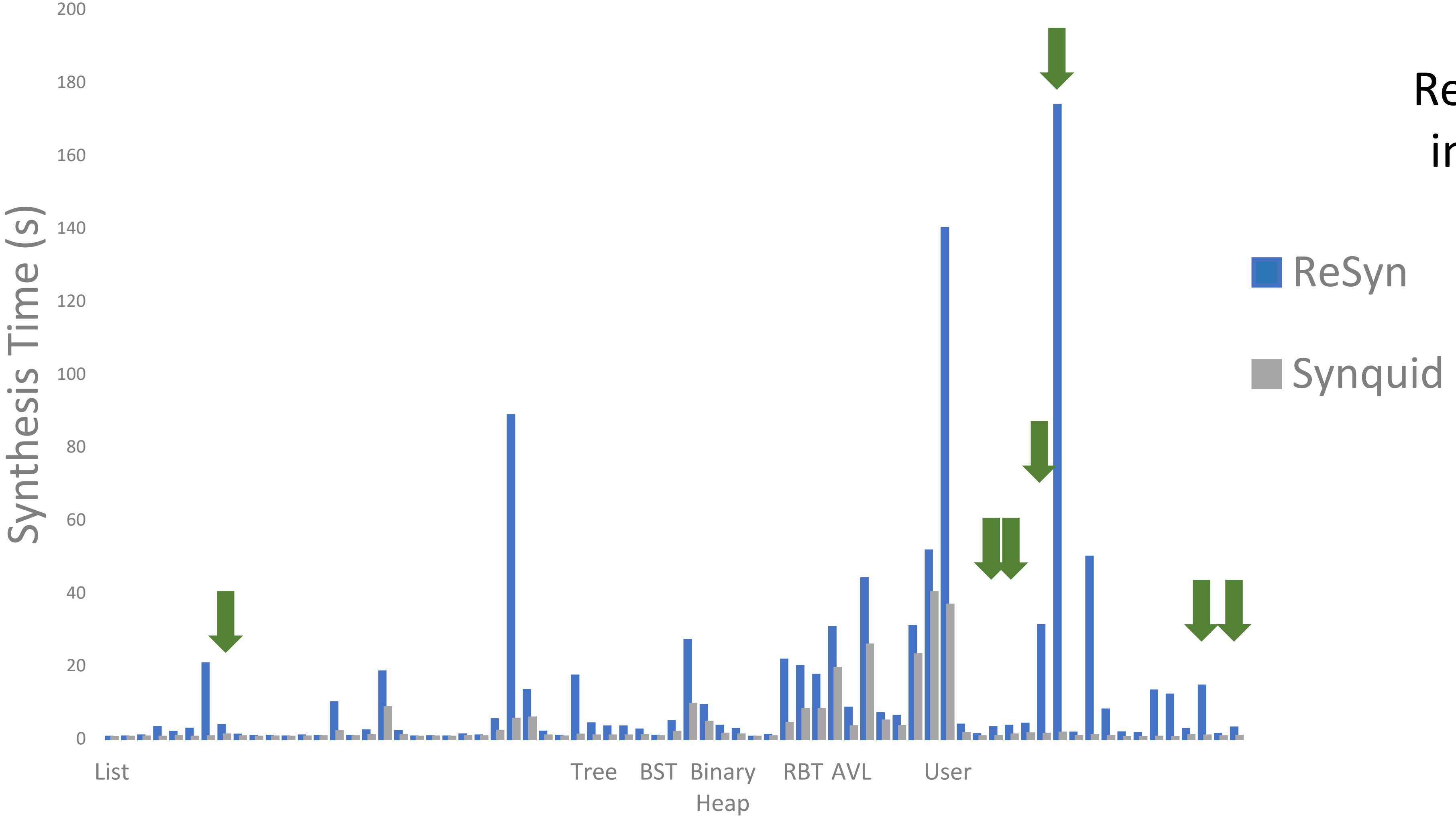
2. How do synthesis times compare?



2. How do synthesis times compare?

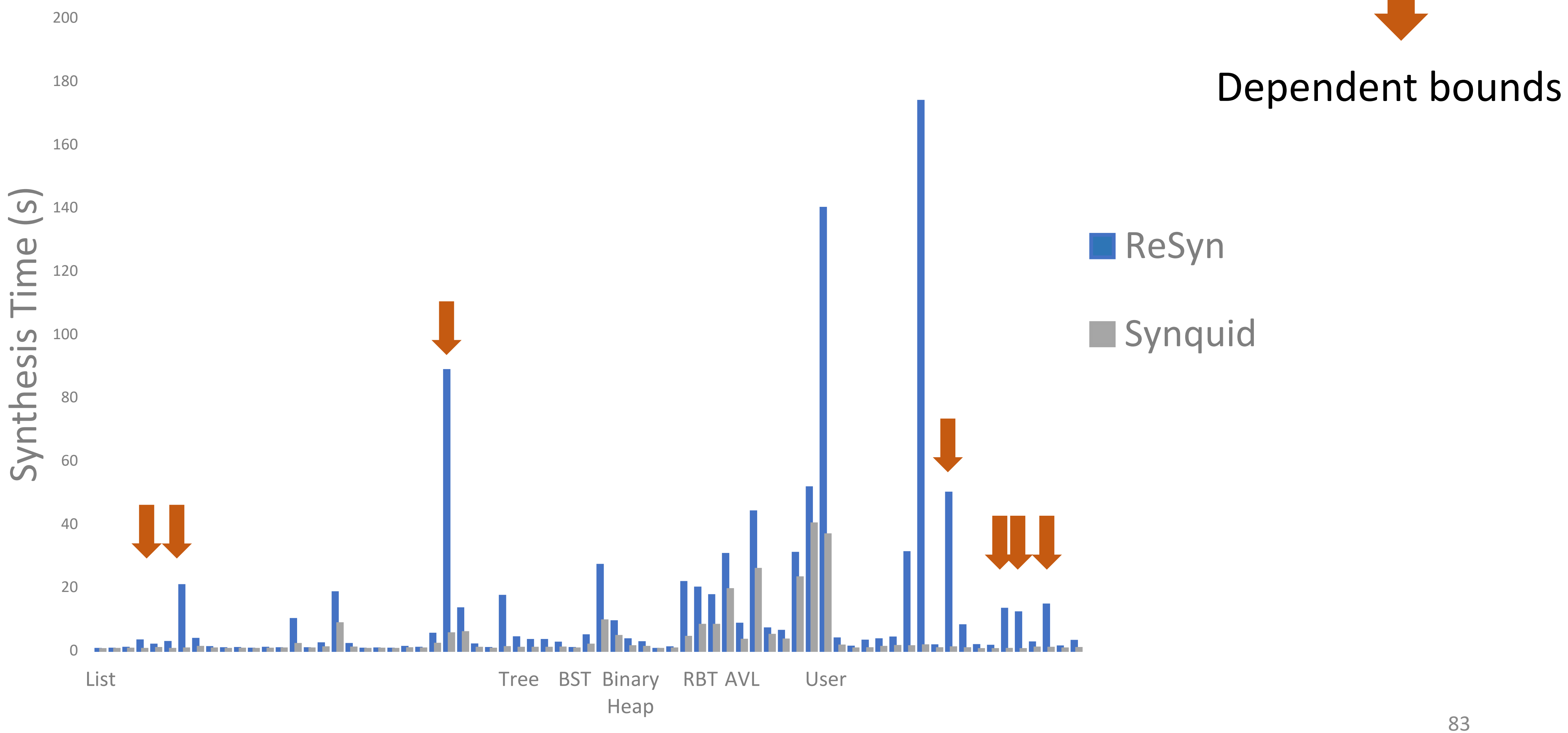


2. How do synthesis times compare?



ReSyn finds faster implementation

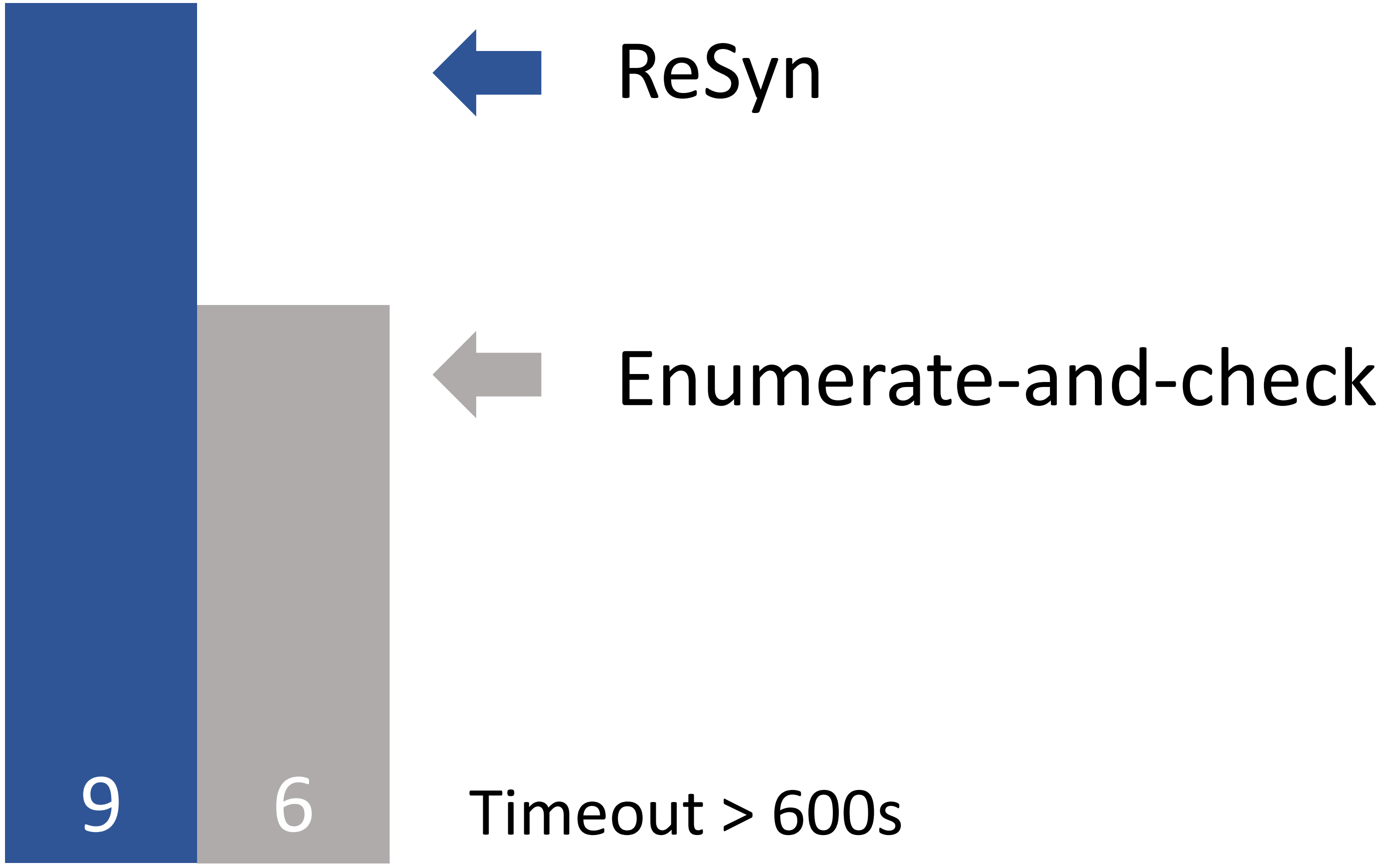
2. How do synthesis times compare?



3. Does local resource analysis guide synthesis?

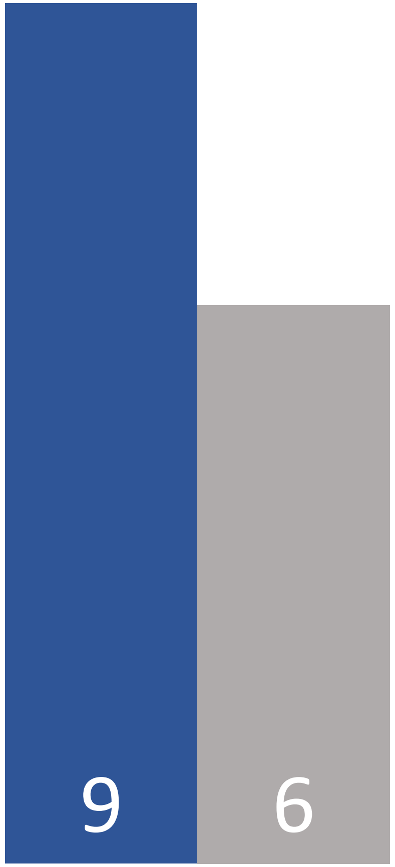
3. What happens if the analysis is non-local?

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



Timeout > 600s

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



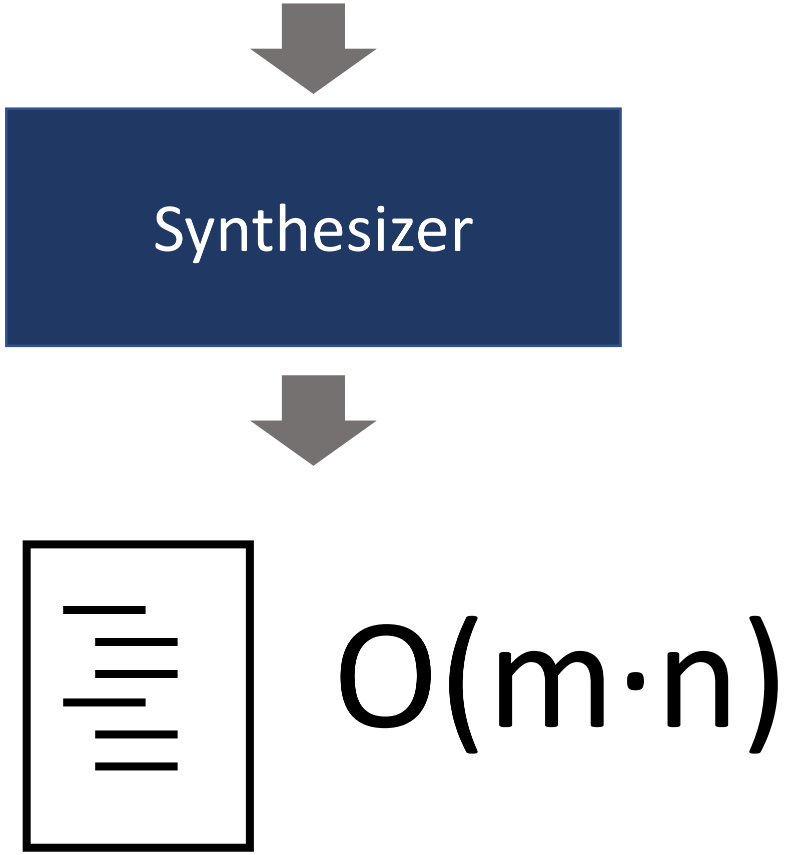
Timeout > 600s

```
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)
```

```
common = λ xs. λ 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)
```

What we had

“Find the intersection of two sorted lists”

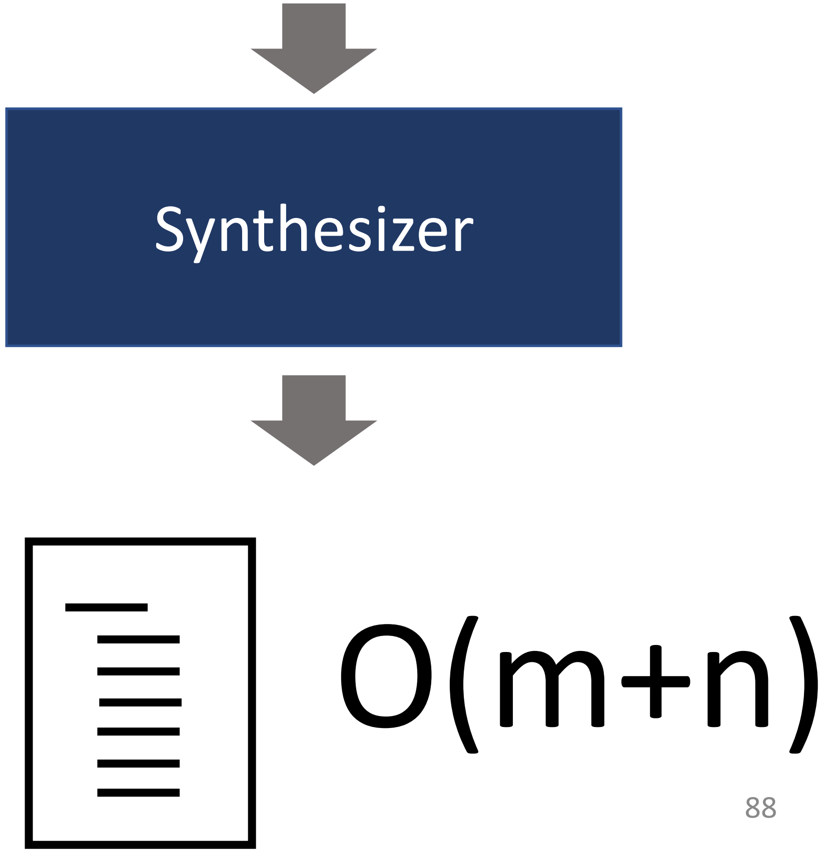


Synthesizer

O(m·n)

What we have now

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



Synthesizer

O(m+n)

https://bitbucket.org/tjknoth/resyn