

Resource-Guided Program Synthesis

Tristan Knoth¹, Di Wang², Nadia Polikarpova¹, Jan Hoffmann²

¹UC San Diego

²Carnegie Mellon University

PLDI 2019

Program Synthesis

Declarative specification



Synthesizer



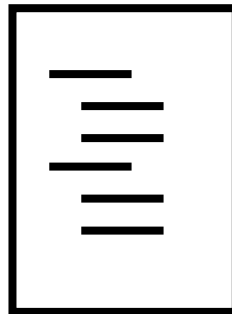
Executable program

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}
```

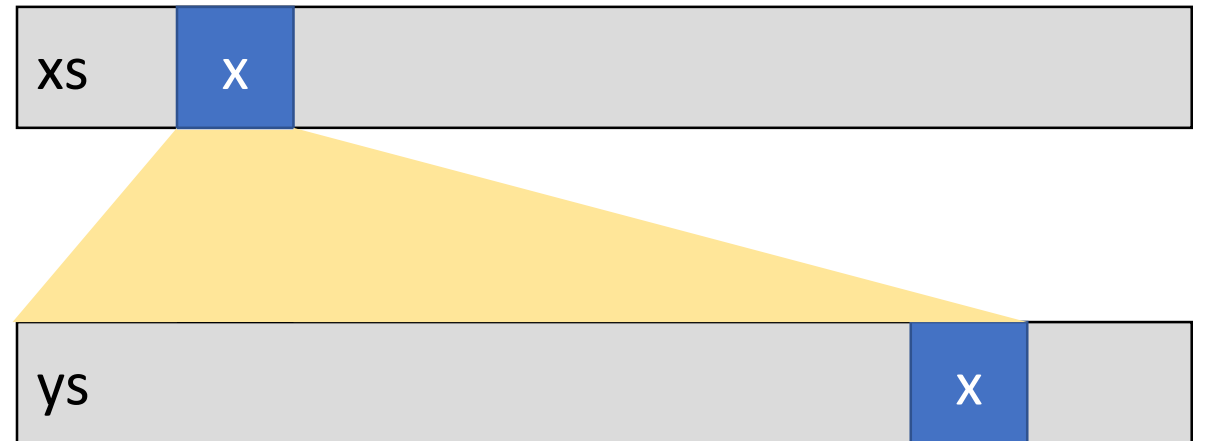


Synthesizer

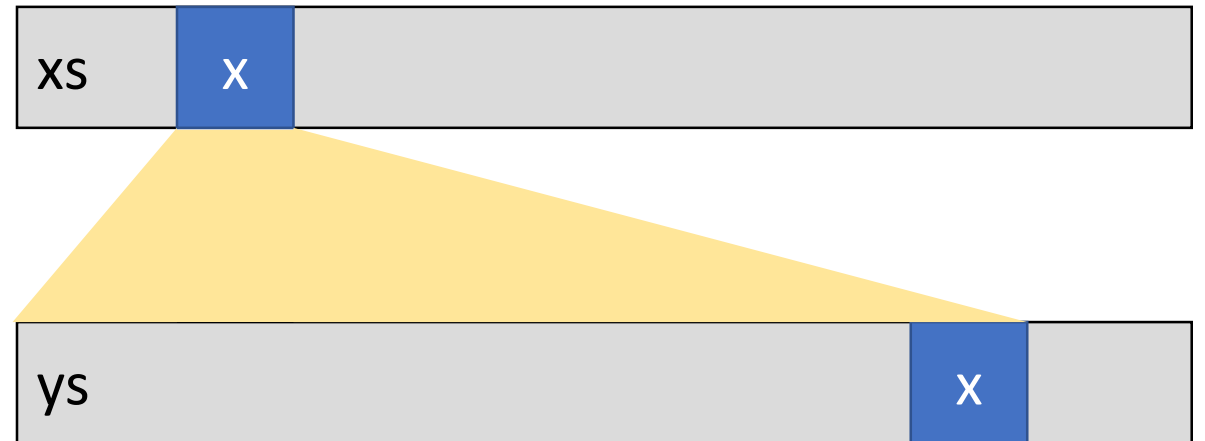


```
common = λ xs. λ ys.  
  match xs with  
  Nil → Nil  
  Cons x xt →  
    if !(member x ys)  
    then common xt ys  
    else Cons x (common xt ys)
```

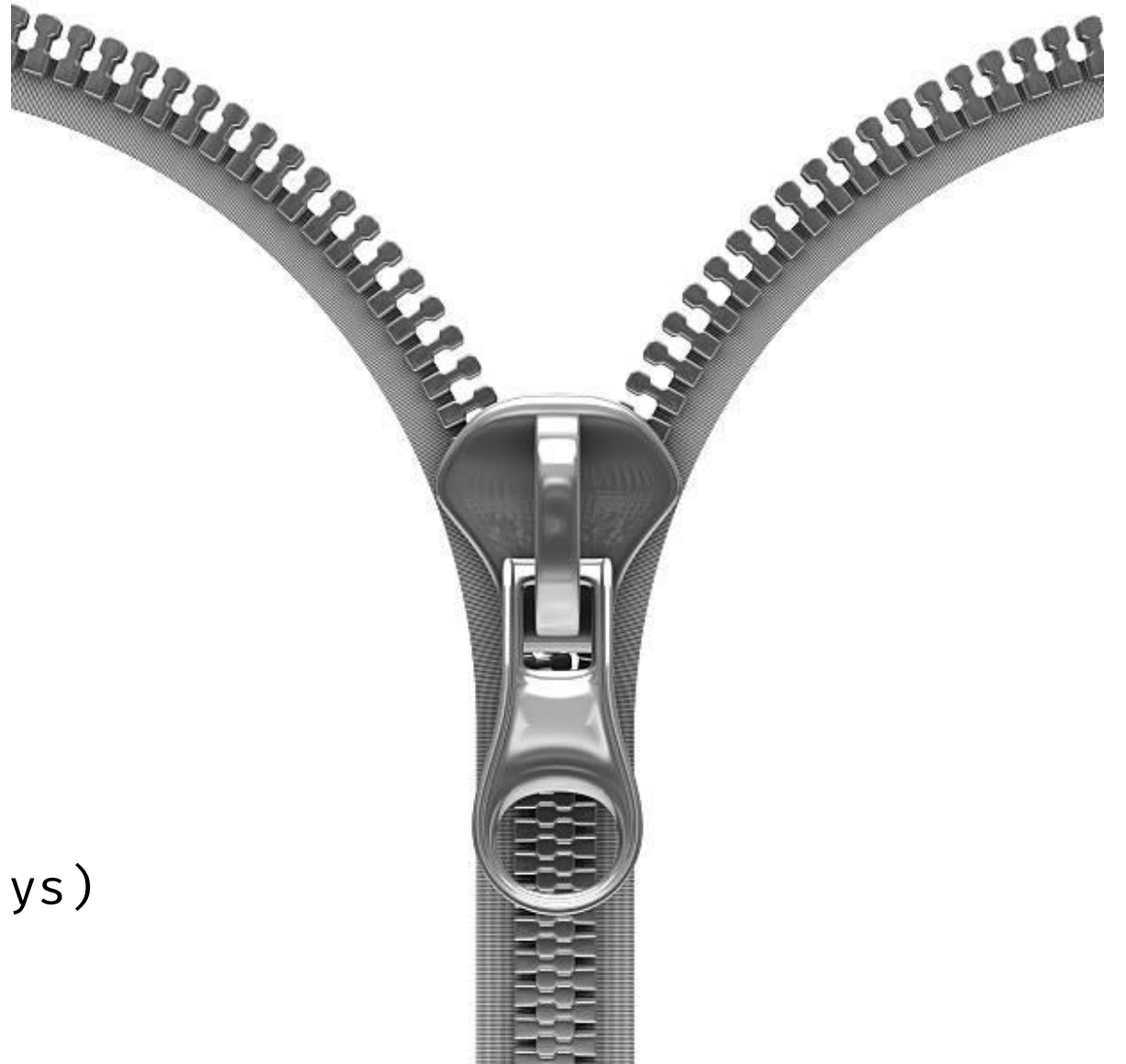
```
common = λ xs. λ ys.  
  match xs with  
    Nil → Nil  
    Cons x xt →  
      if !(member x ys)  
      then common xt ys  
      else Cons x (common xt ys)
```



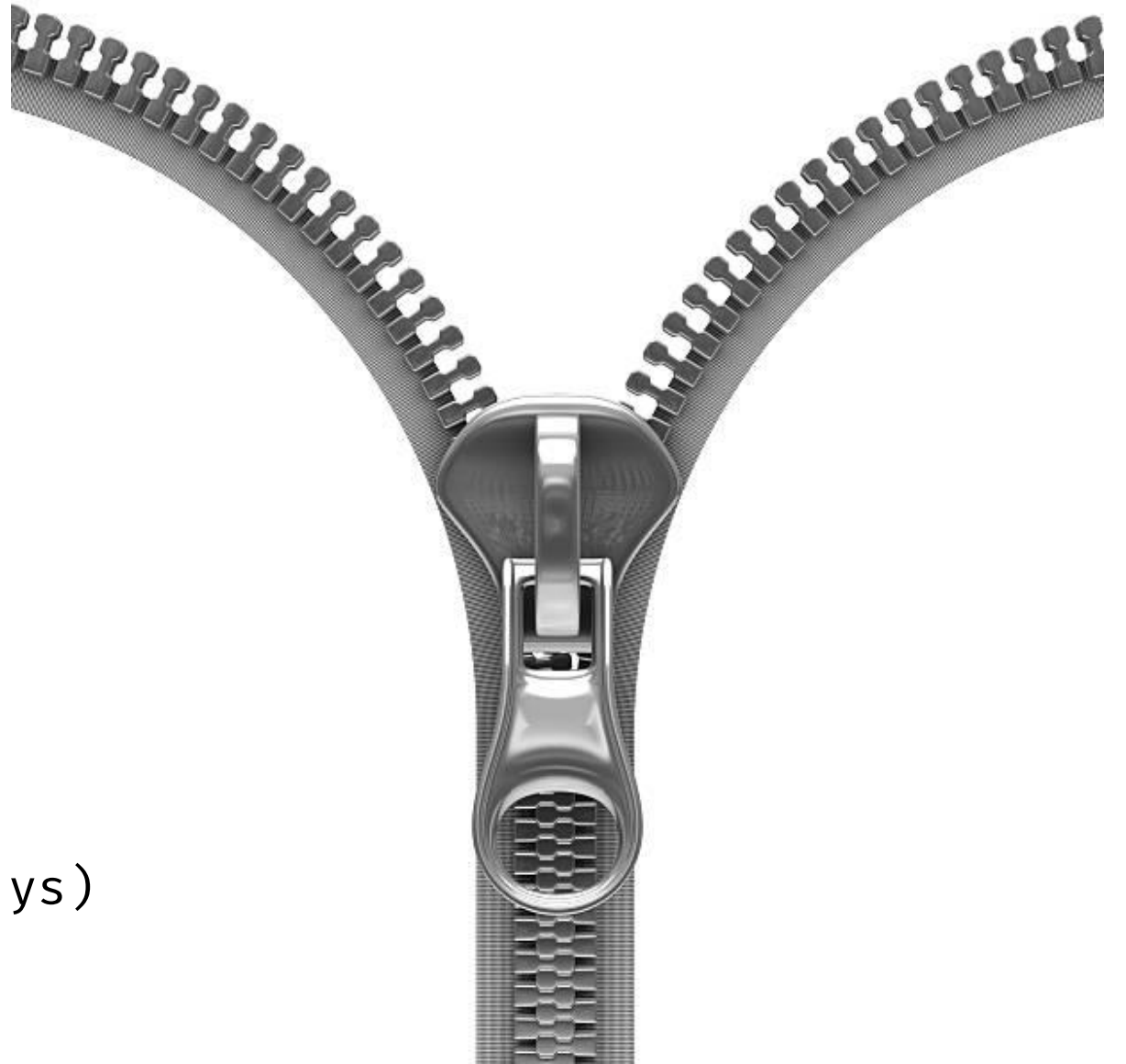
```
common = λ xs. λ ys.  
  match xs with  
    Nil → Nil  
    Cons x xt →  
      if !(member x ys)  
      then common xt ys  
      else Cons x (common xt ys)
```



```
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 yt)
```



$O(m \cdot n)$

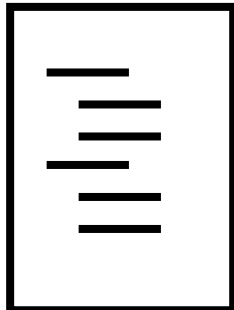
```
common = λ xs. λ ys.  
  match xs with  
    Nil → Nil  
    Cons x xt →  
      if !(member x ys)  
        then common xt ys  
        else Cons x (common xt ys)
```

$O(m + n)$

```
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

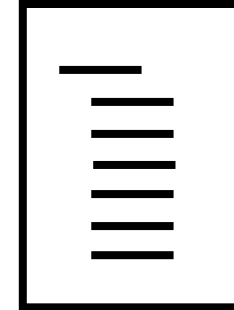
“Find the intersection of two sorted lists”



$O(m \cdot n)$

What we want

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



$O(m+n)$

ReSyn

The first resource-aware
synthesizer for recursive programs

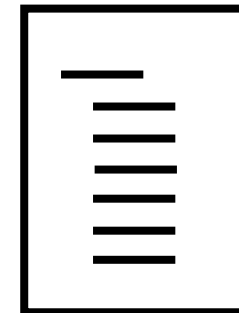
This talk

1. Specification

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



Synthesizer



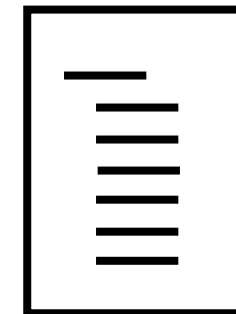
This talk

1. Specification

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



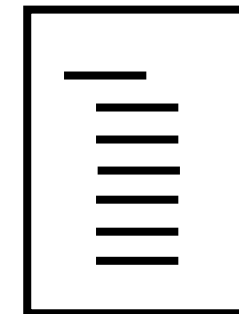
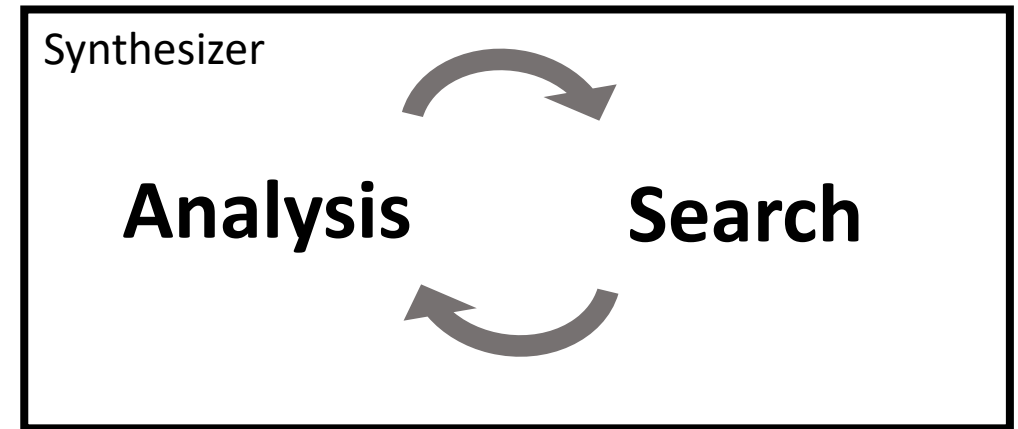
Synthesizer



This talk

1. Specification

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

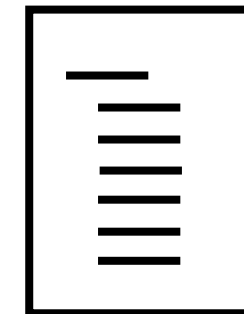
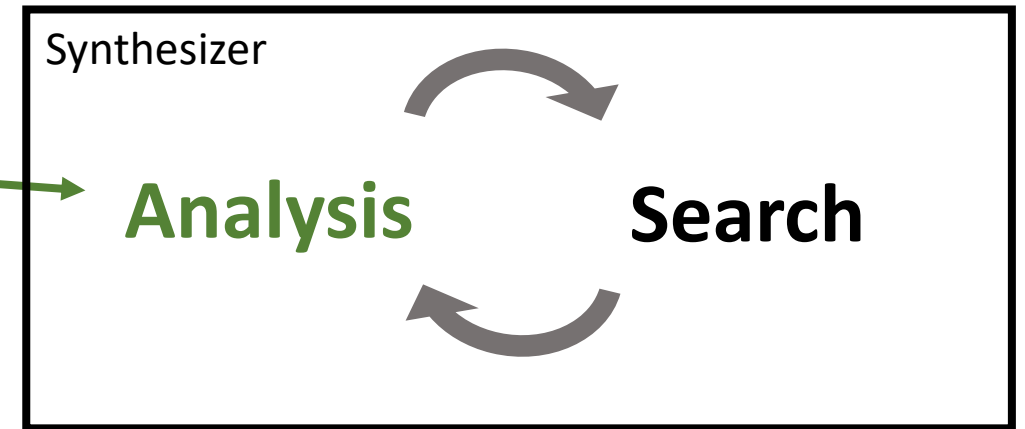


This talk

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

1. Specification

2. Analysis



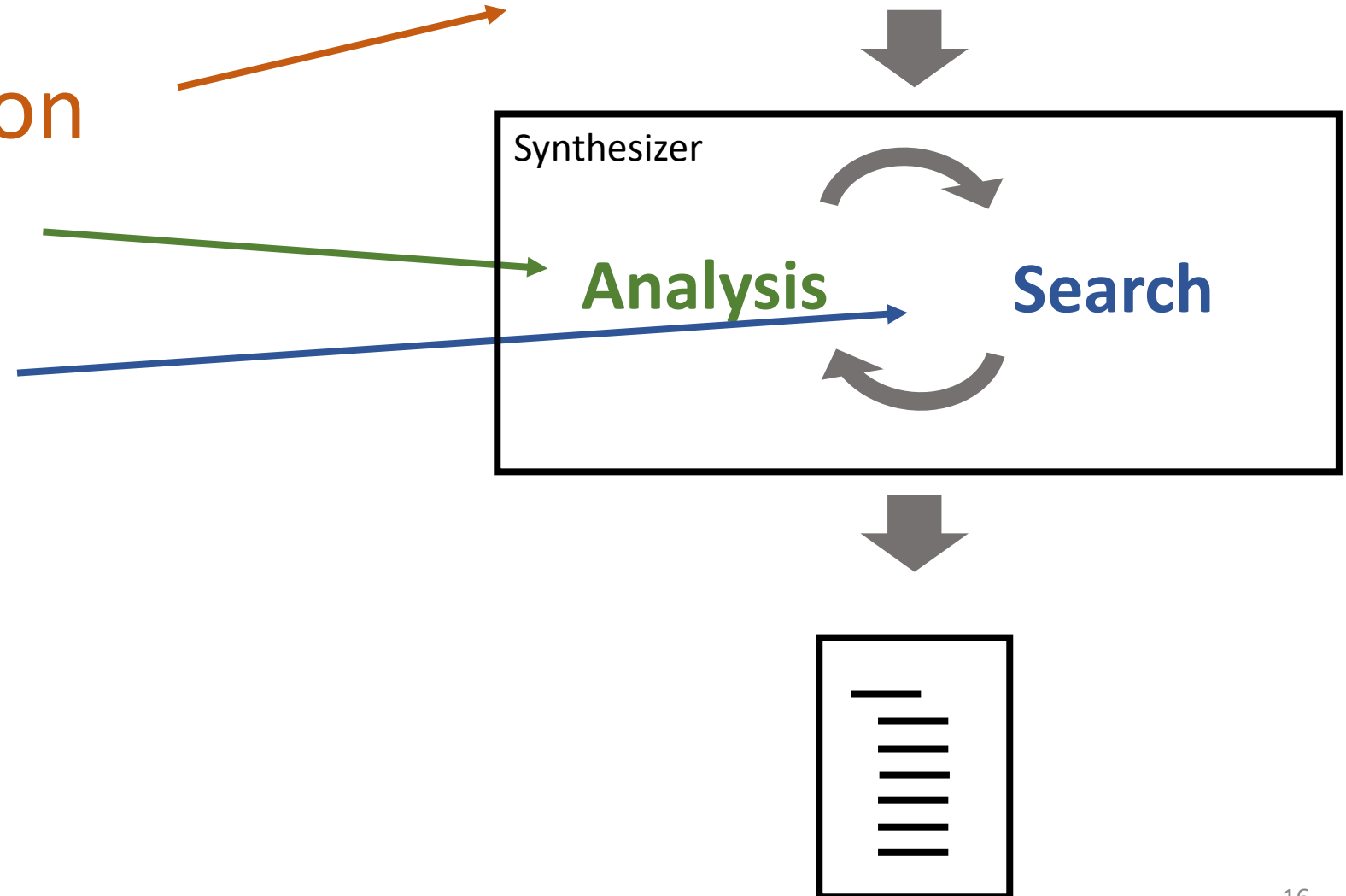
This talk

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”



??

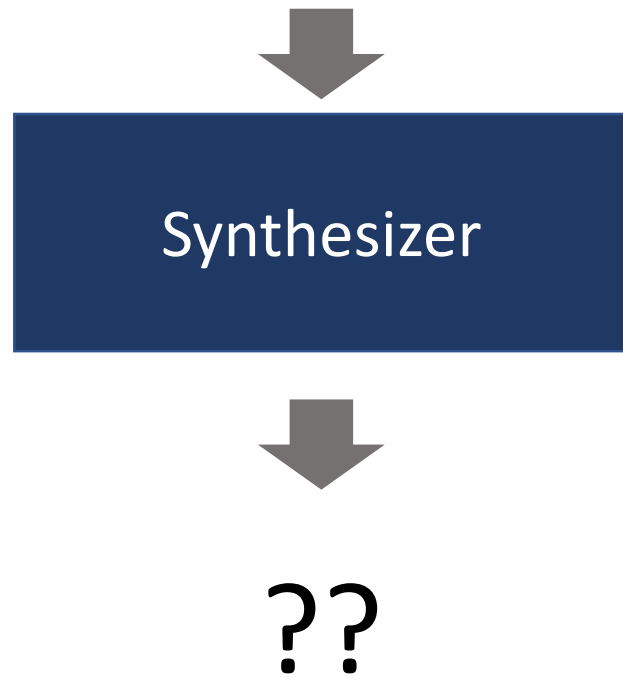
“Find the intersection of two sorted lists in linear time”



??

Refinement types

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



Refinement types
with
Resource annotations

Refinements: Synquid



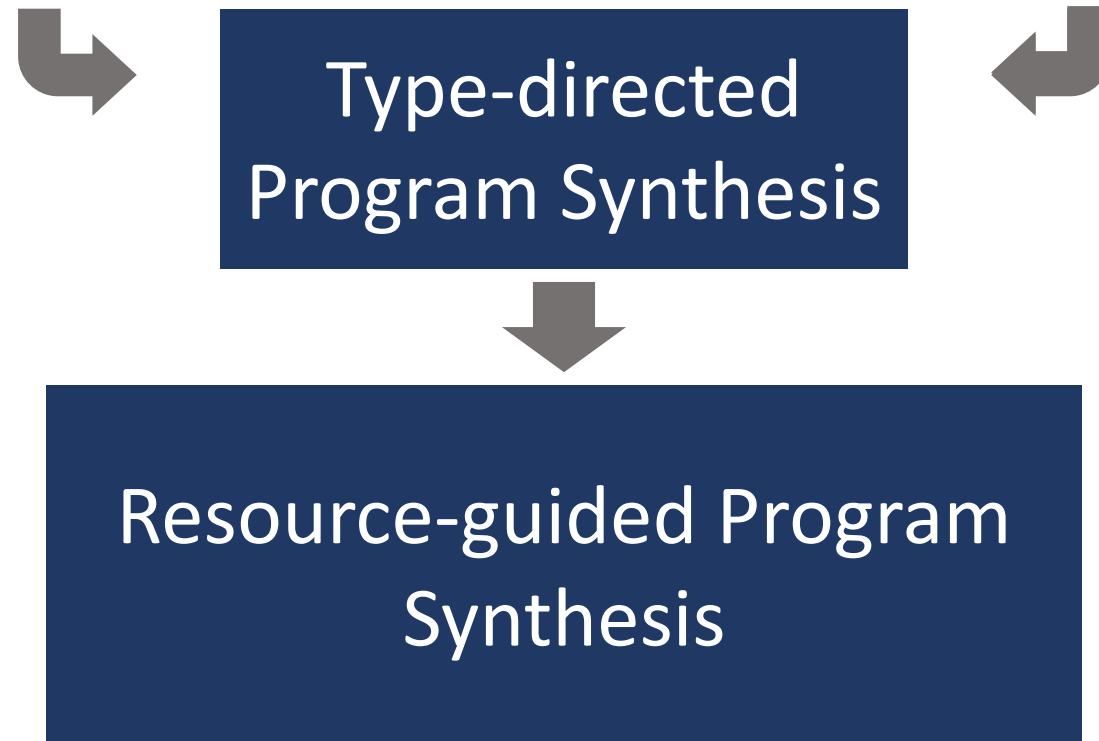
Type-directed
Program Synthesis



Resource-guided Program
Synthesis

Refinements: Synquid

Resource annotations: AARA



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

{ B | Ψ }

$v : \{ \text{Int} \mid v \geq 0 \}$

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 n elems ys}  
common = ??
```

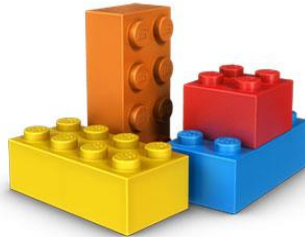
Refinement types

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

**Functional
specification**



Library
functions



[Polikarpova et. al, 2016]

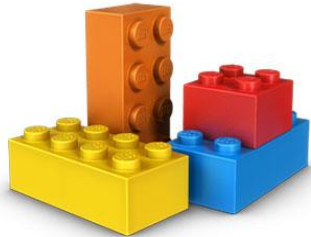
Functional
specification



```
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**”

Potential



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

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

Potential: numeric



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



Refinement: boolean

Resource annotations

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

Resource budget

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

Synthesize with ReSyn

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

member
Cons, Nil, ...
≤, =, !, ...



Components: `member`

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

Components: `member`

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

Components: `member`

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

**Functional
specification**

Resource bound

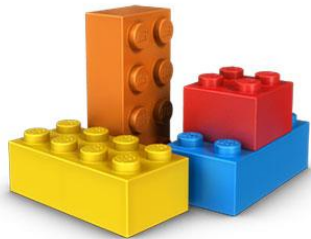
Library
functions



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
```

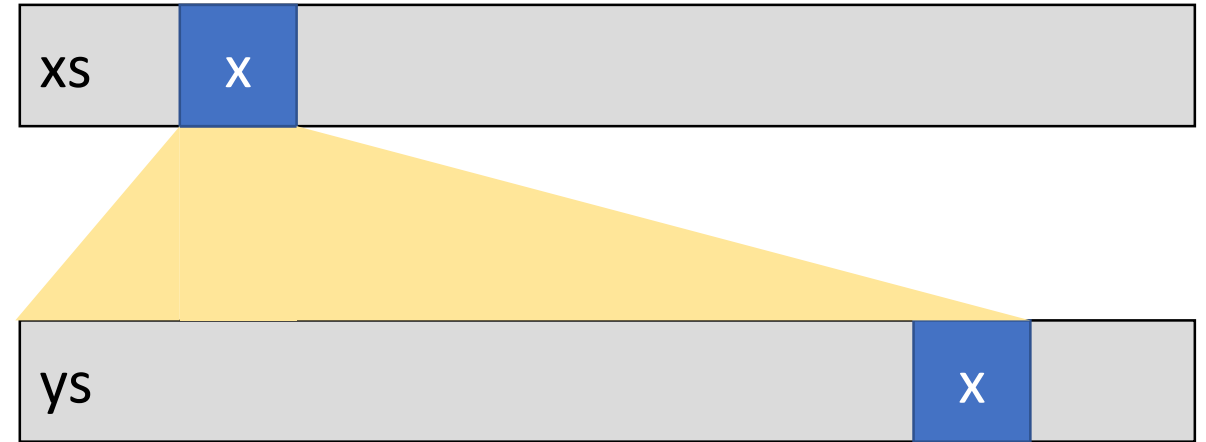
```
    Nil → Nil
```

```
    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 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  
        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)
```

ys :: SList a¹

```
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 ap))  
        then common xt (ys :: List aq)  
        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 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.                                List ap <: List a1
  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.                                List ap <: List a1
  match xs with                                       p ≥ 1
    Nil → Nil
    Cons x xt →
      if !(member x (ys :: List ap))
      then common xt ys
      else Cons x (common xt ys)
```

$$a <: b \qquad p \geq q$$

$$\text{List } a^p <: \text{List } b^q$$

`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 aq)  
        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)`

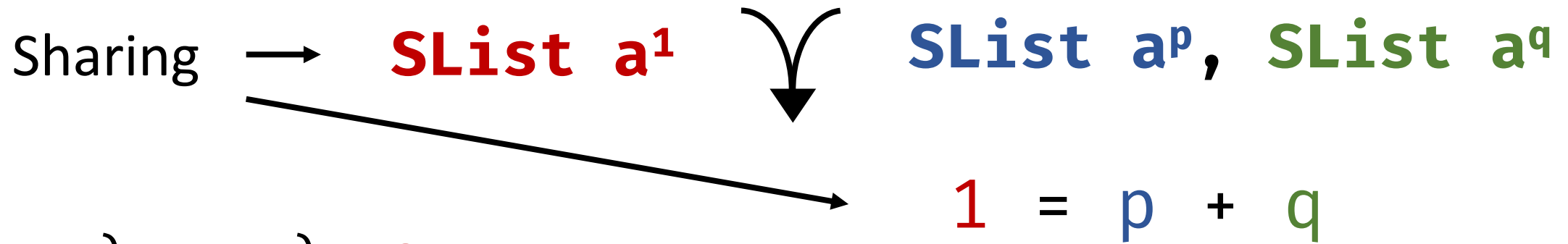
`then common xt (ys :: List aq)`

`else Cons x (common xt ys)`

`List aq <: List a1`
`q ≥ 1`

Sharing \rightarrow **SList** a^1  **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)
```



```
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)
```

Sharing \rightarrow **SList** a^1 \searrow **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)
```

$$\begin{array}{rcl} 1 & = & p + q \\ p & \geq & 1 \\ q & \geq & 1 \end{array}$$

Subtyping

SMT

$$\begin{aligned} 1 &= p + q \ \&\& \\ p &\geq 1 \ \&\& \\ q &\geq 1 \end{aligned}$$

SMT

$$\begin{aligned} 1 &= p + q \ \&\& \\ p &\geq 1 \ \&\& \\ q &\geq 1 \end{aligned}$$



This talk

1. Specification
2. Analysis
3. Search

Synthesis is search

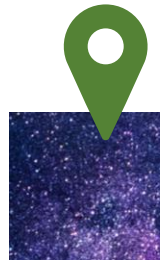


Synthesis is search



Synthesis is search

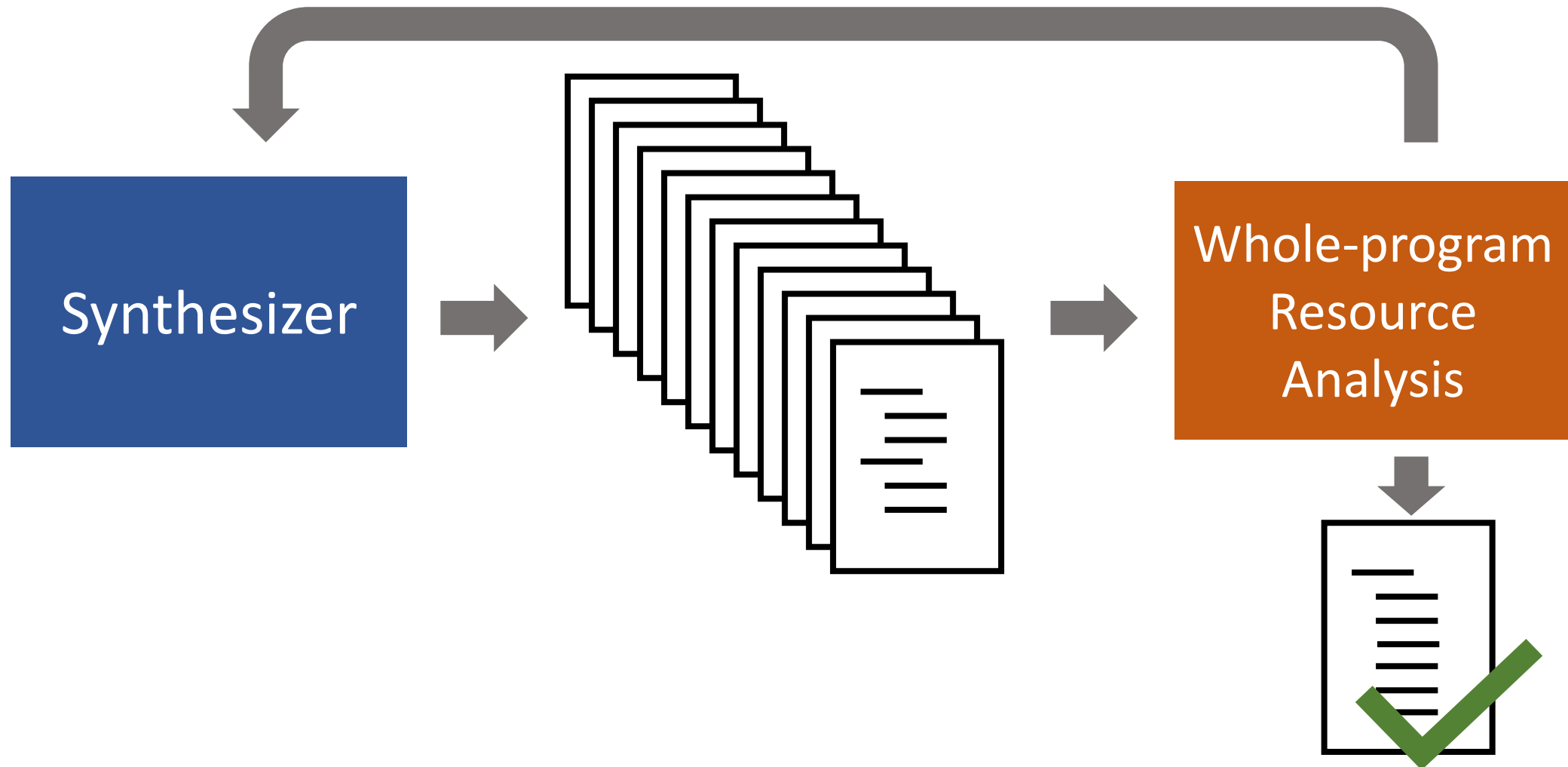
Synthesis is search



Reject impossible programs early

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

A different approach

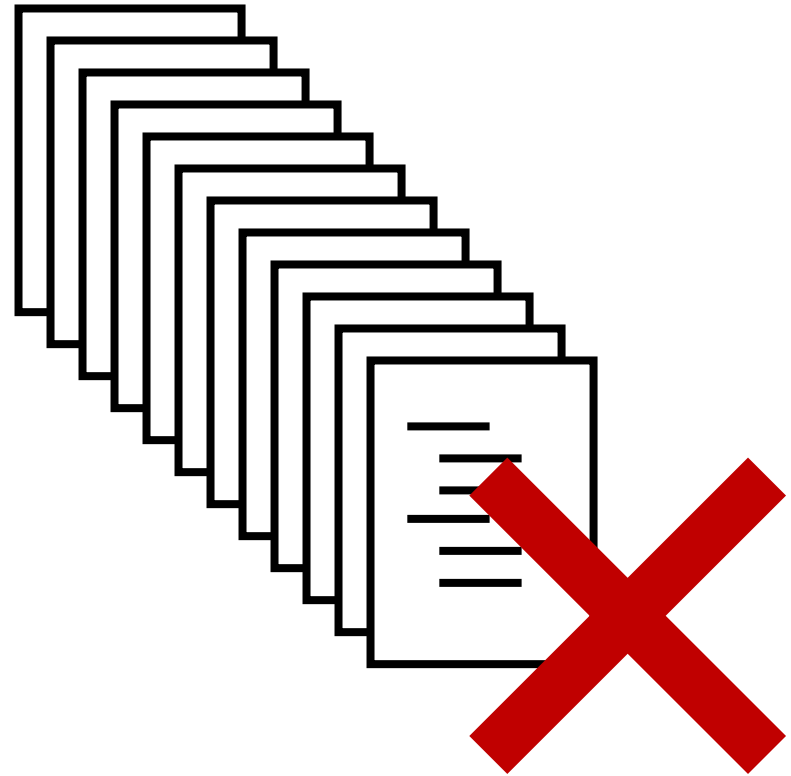
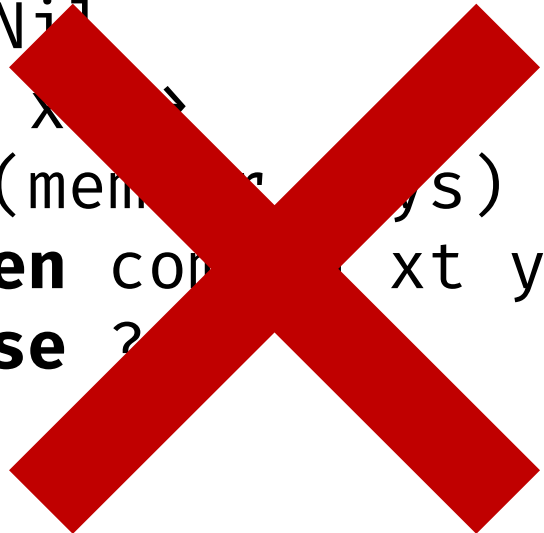


Reject impossible programs early with local analysis

```
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 xs' →  
      if !(member x ys)  
      then common xs' 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 ??
```

Results

Results

1. Can ReSyn generate faster programs than Synquid?

Results

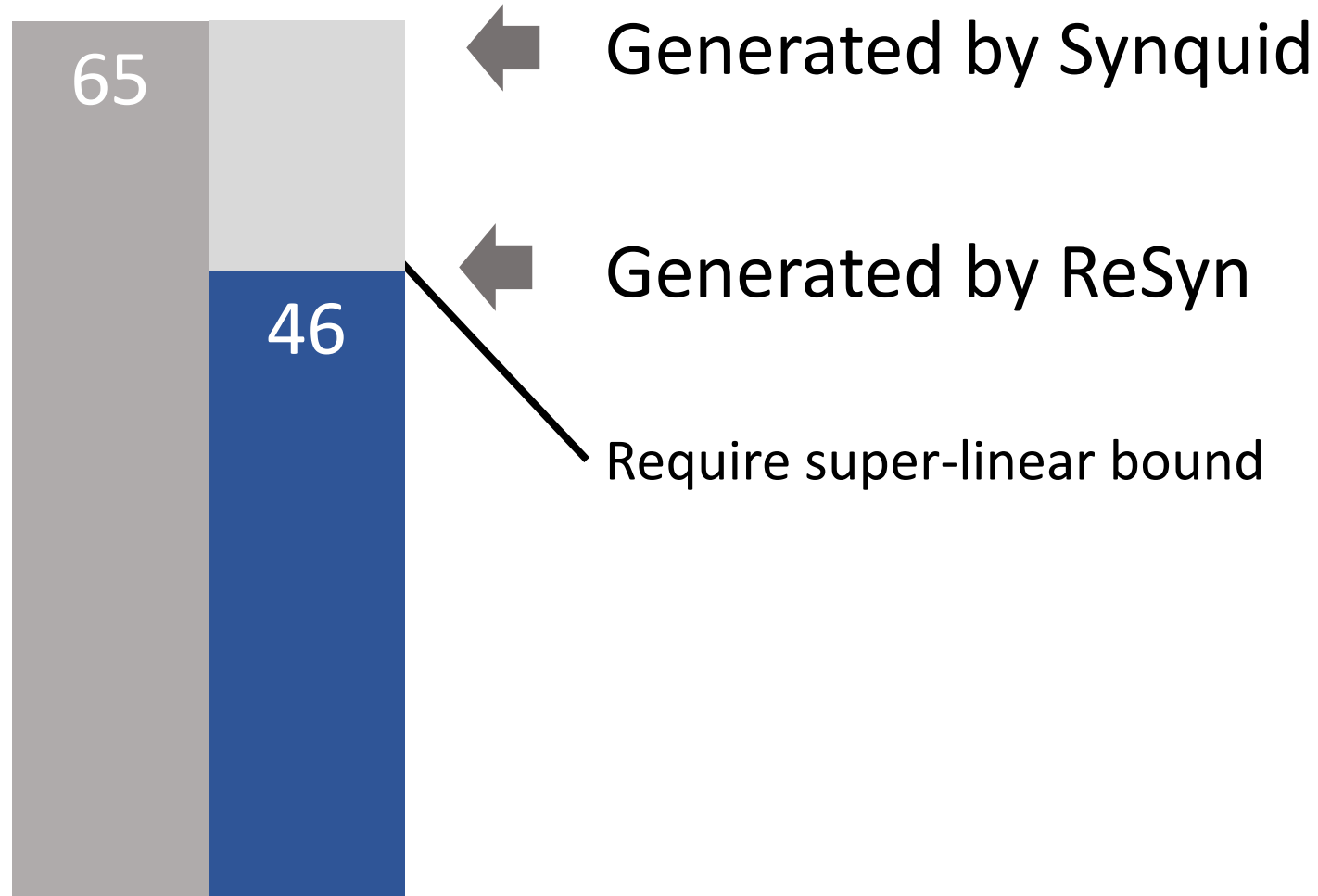
1. Can ReSyn generate faster programs than Synquid?
2. How much longer does ReSyn take to generate code?

Results

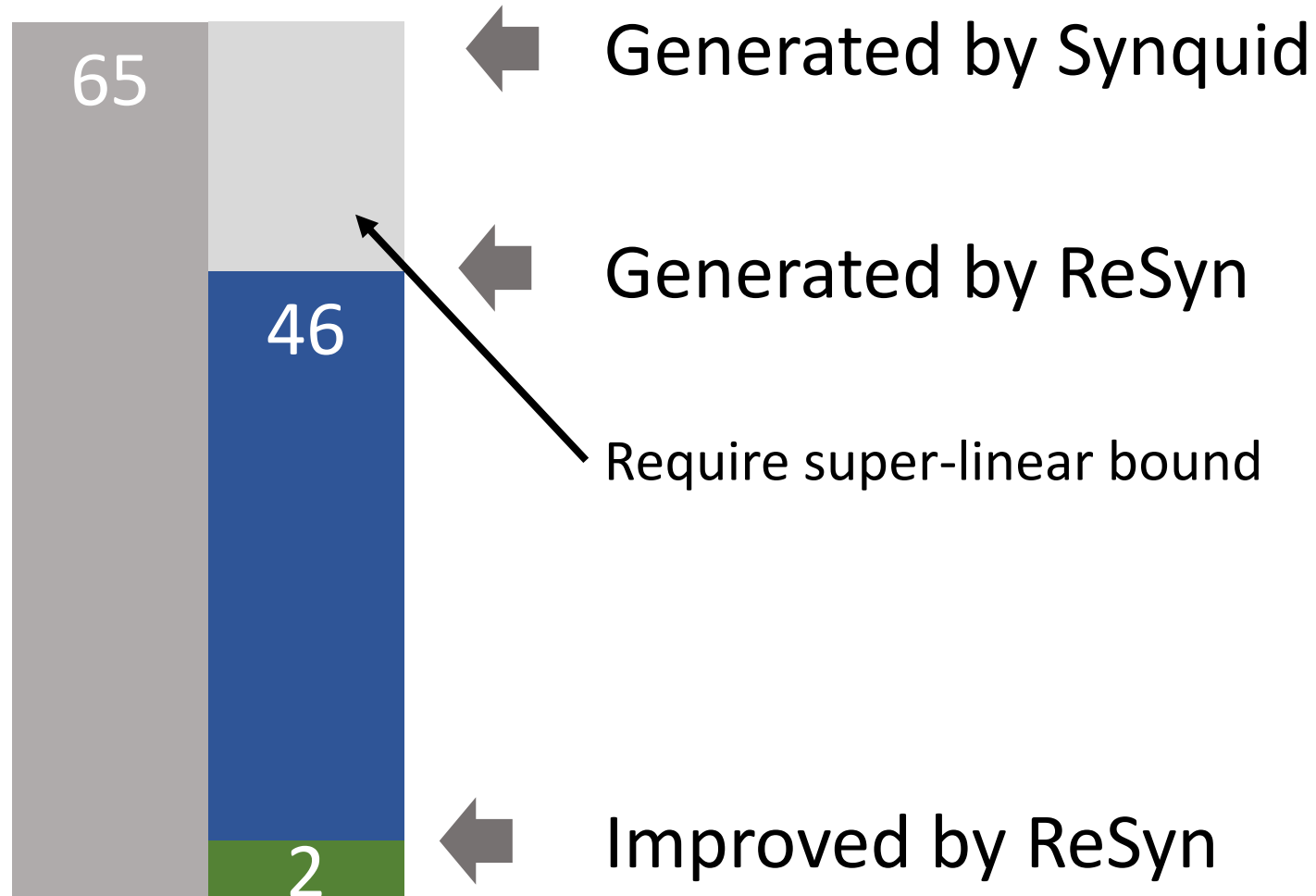
1. Can ReSyn generate faster programs than Synquid?
2. How much longer does ReSyn take to generate code?
3. Is local resource analysis effective at guiding the search?

1. Can ReSyn generate faster programs?

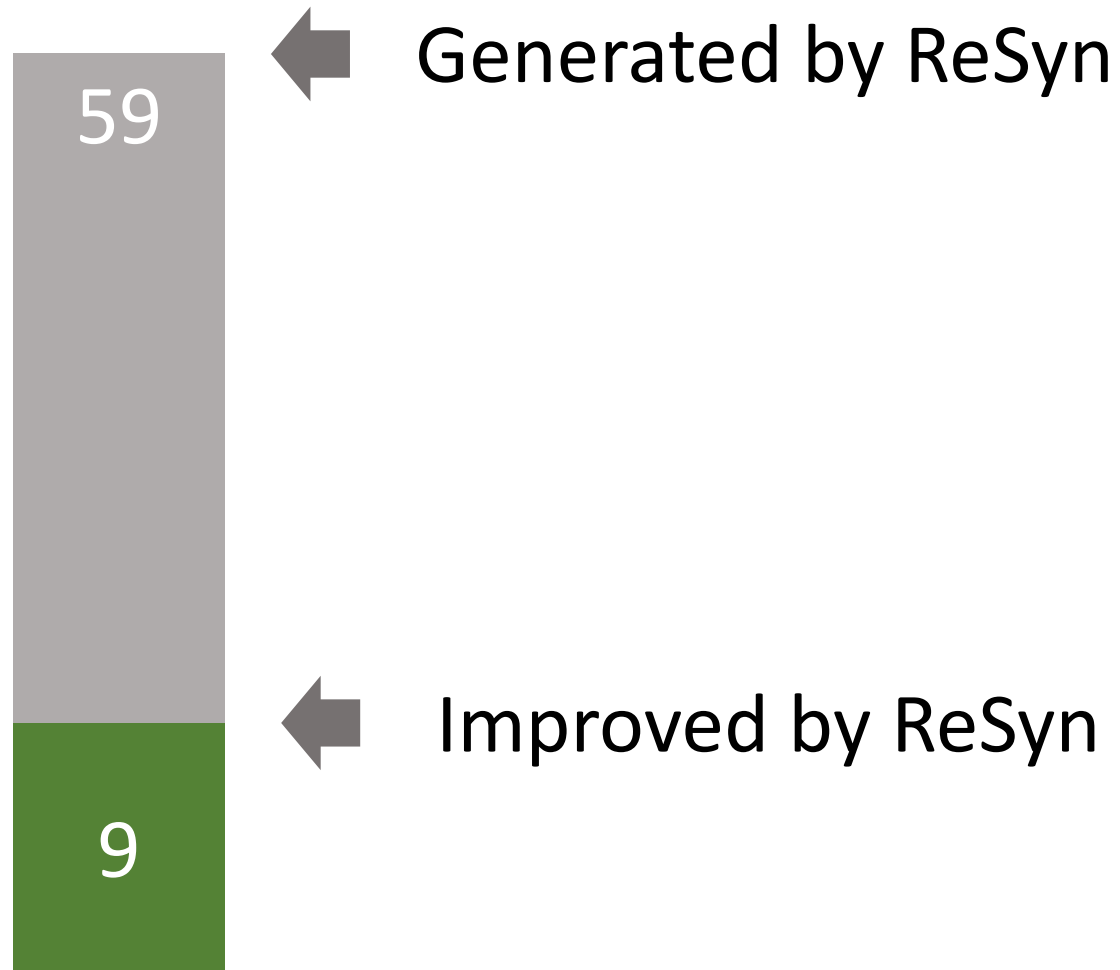
1. Can ReSyn generate faster programs?



1. Can ReSyn generate faster programs?



1. Can ReSyn generate faster programs?

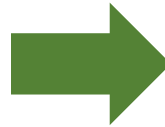


compress: Remove adjacent duplicates

```
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)
```

$O(2^n)$

Synquid



```
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)
```

$O(n)$

ReSyn



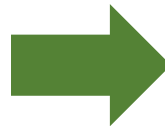
insert: Insert into a sorted list

```
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

`insert :: x:a → xs: SList a`
`if x > v then 1 else 0`
`→ 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)

```

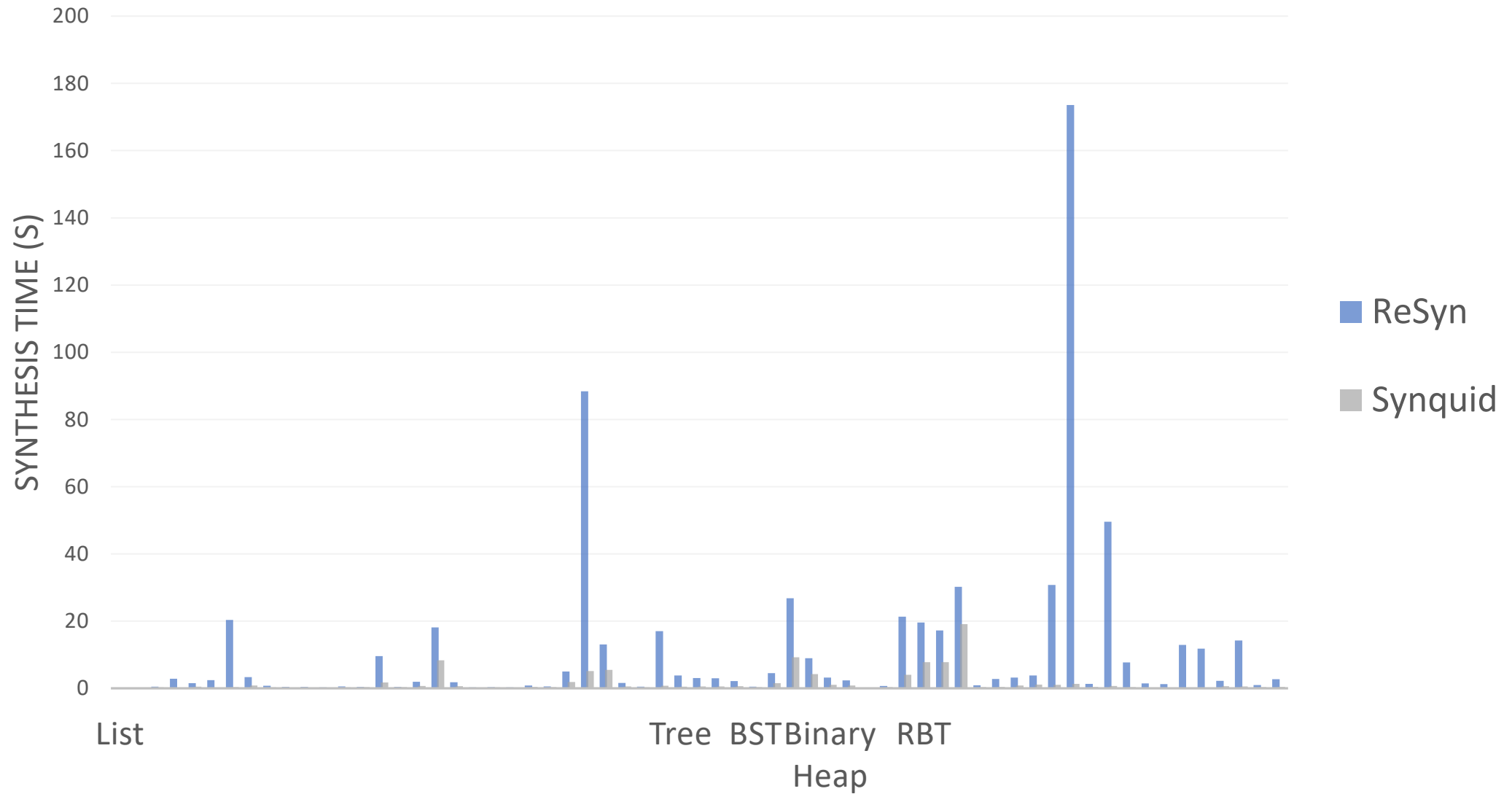
$O(n)$



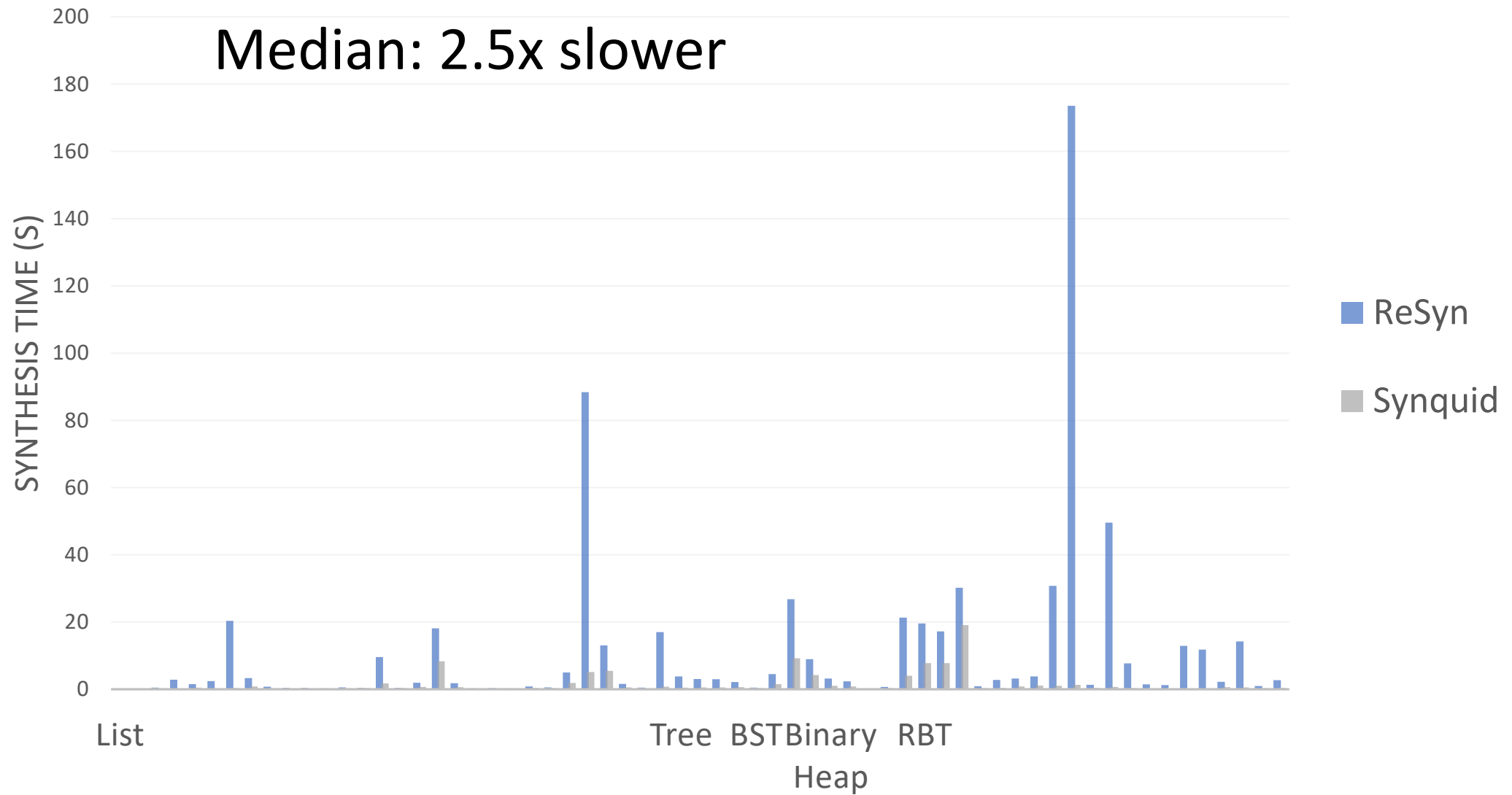
$O(n)$

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

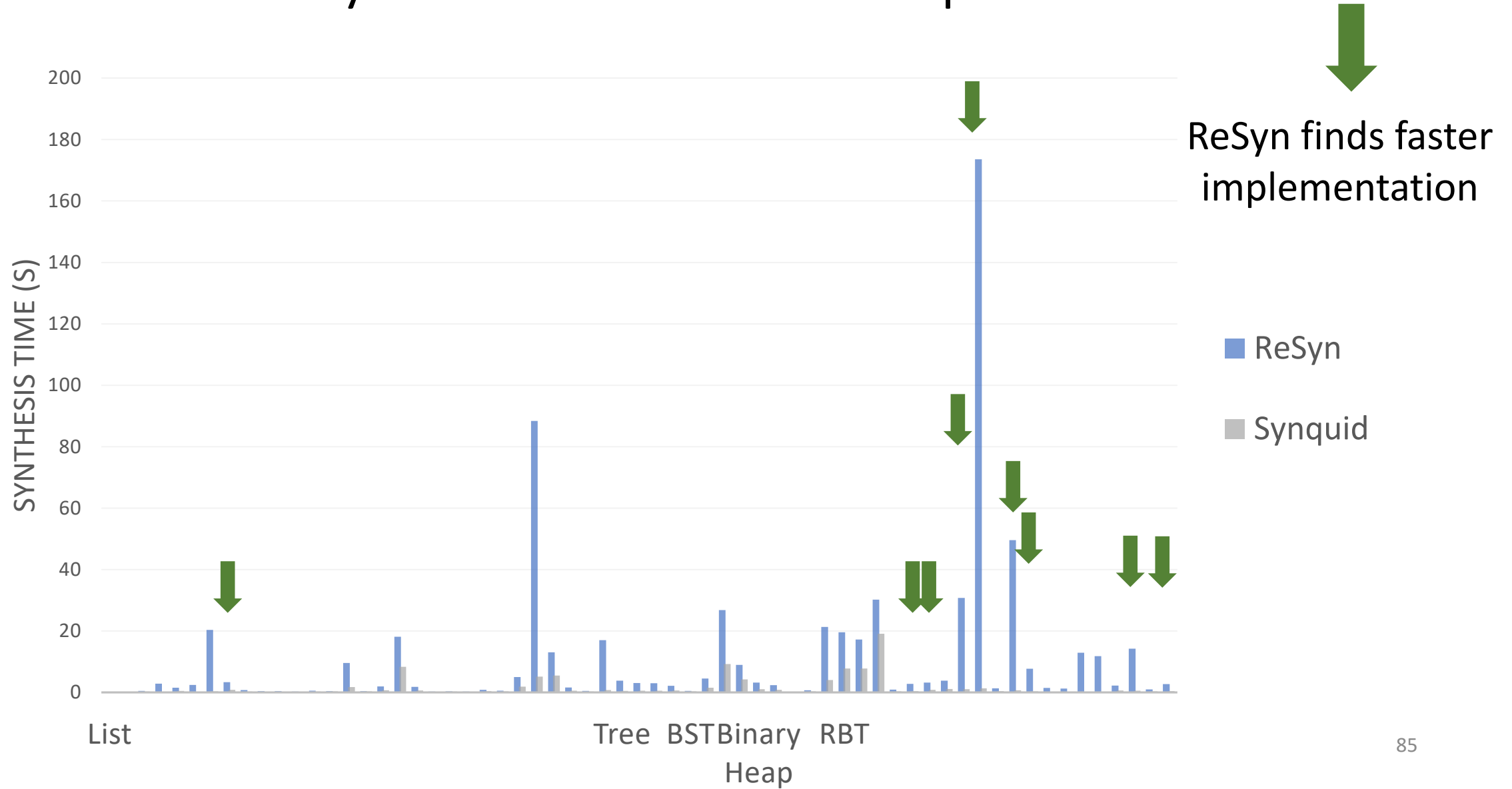
2. How do synthesis times compare?



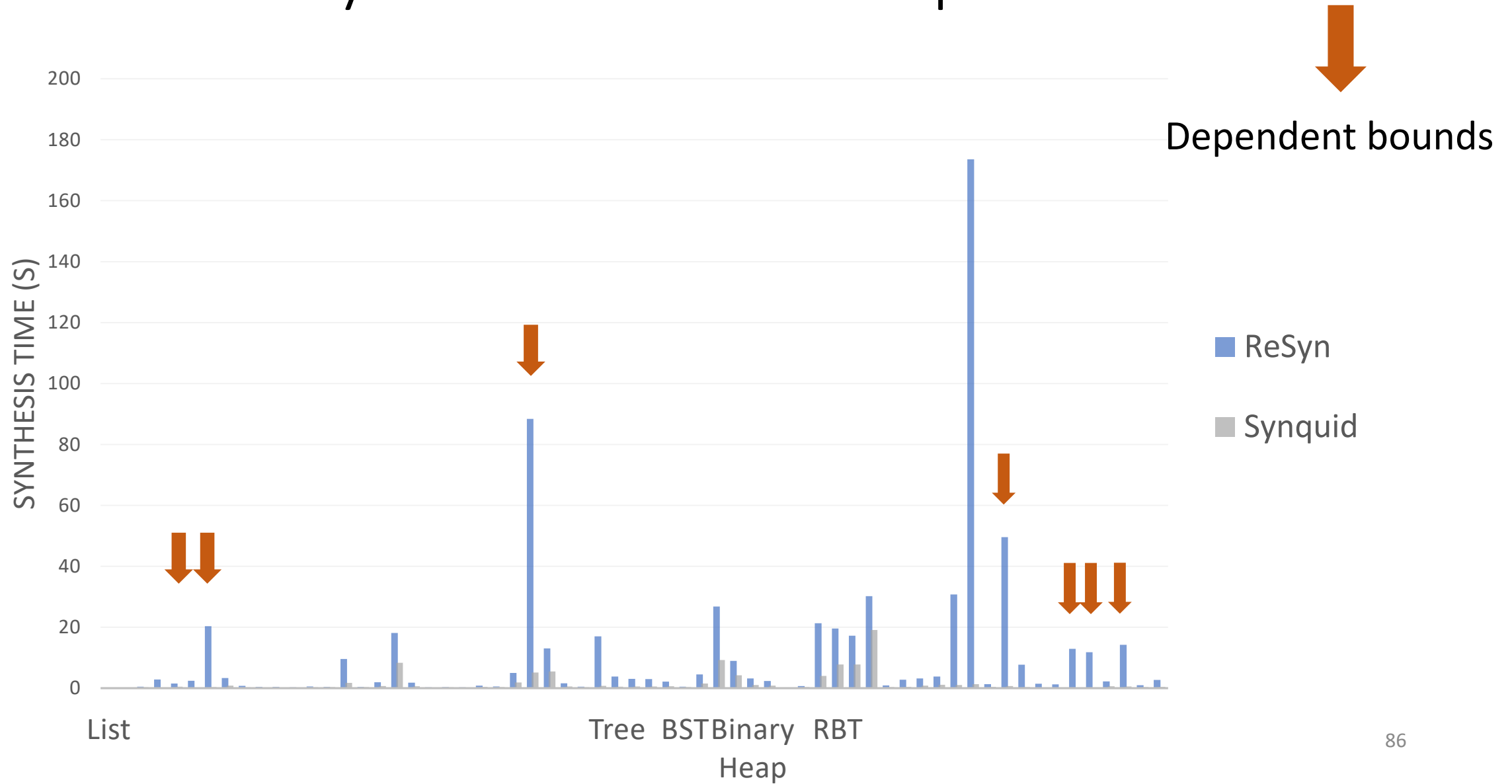
2. How do synthesis times compare?



2. How do synthesis times compare?



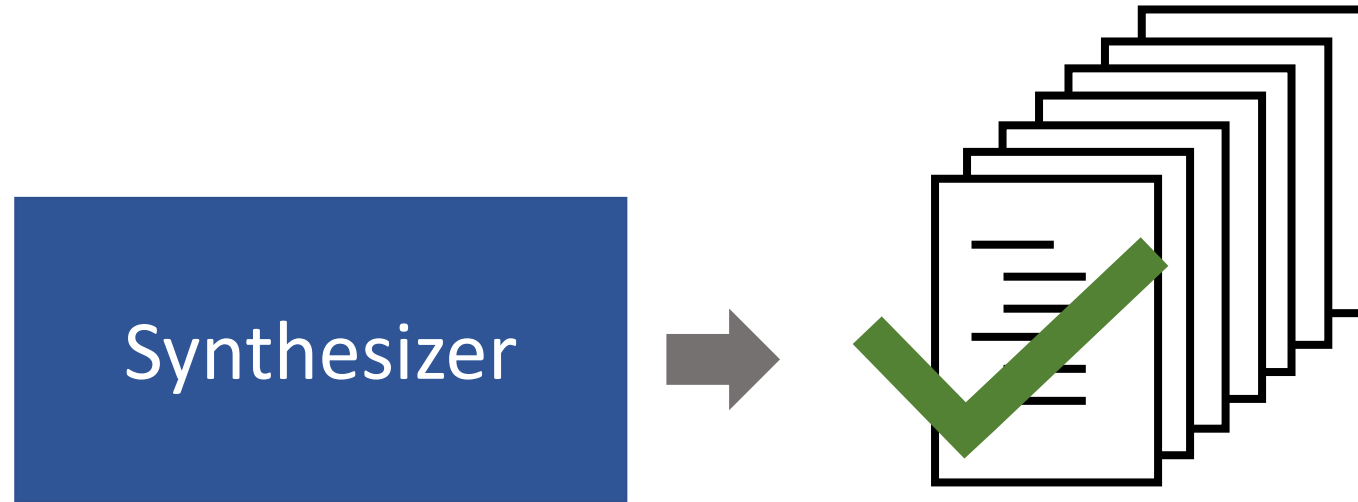
2. How do synthesis times compare?



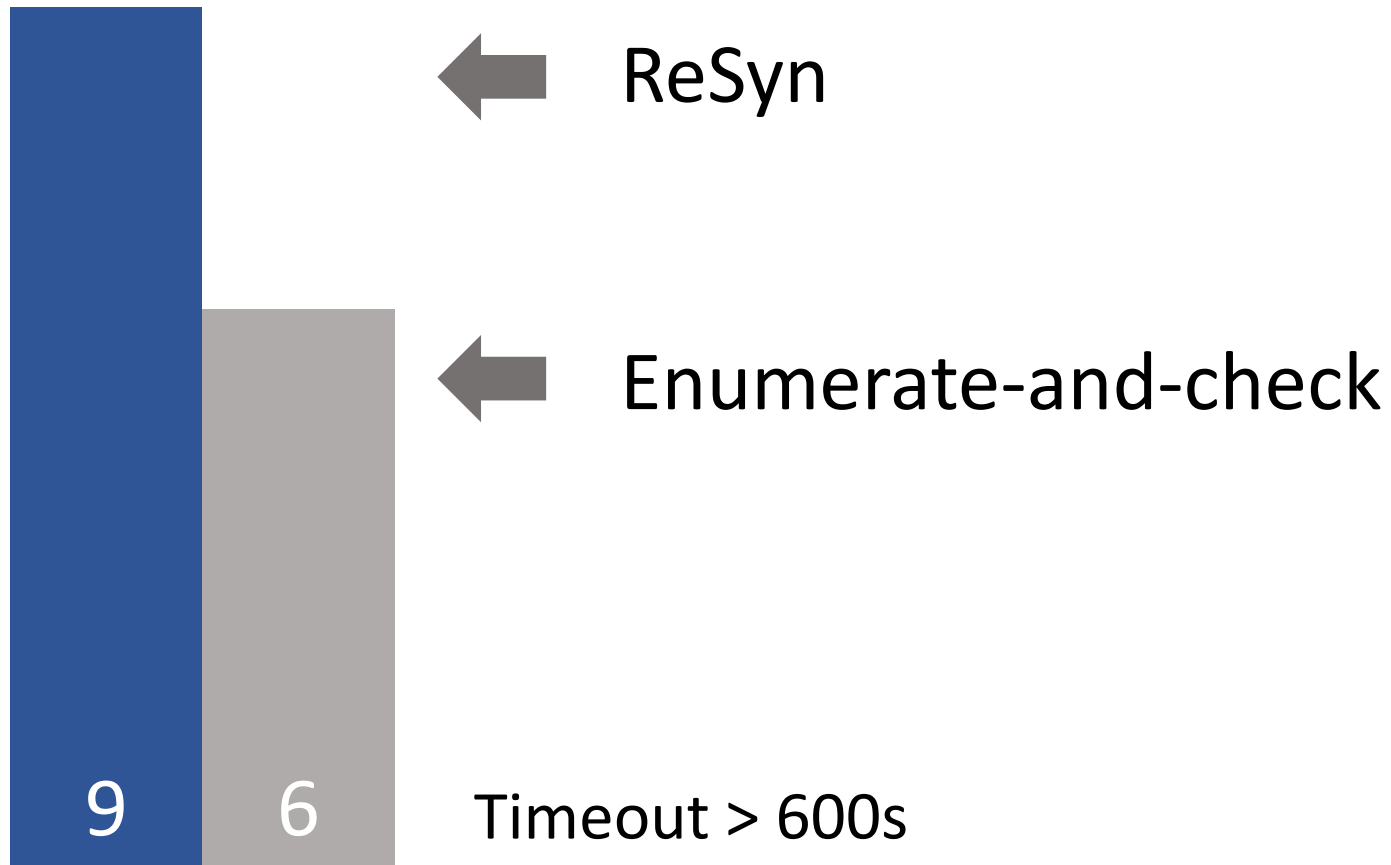
3. Does local resource analysis guide synthesis?

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

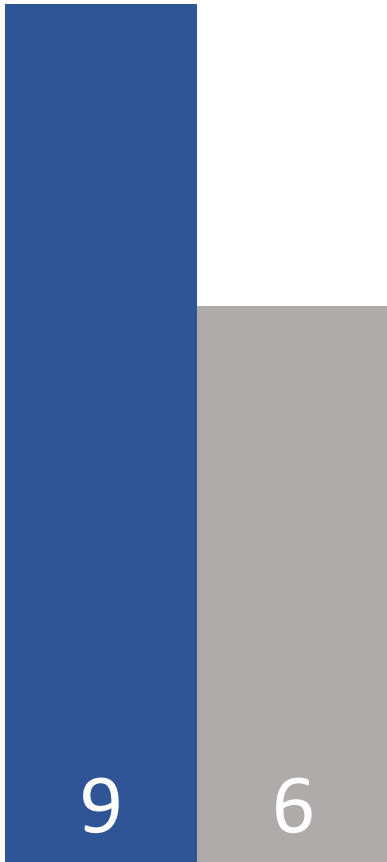
3. What happens if the analysis is non-local?



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



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



Timeout > 600s

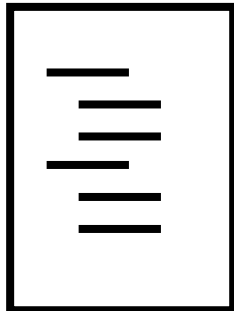
```
common = λ xs. λ ys.  
  match xs with  
  Nil → Nil  
  Cons x xt →  
    if !(member x ys)  
    then common xt ys  
    else Cons x (common x ys)
```



```
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

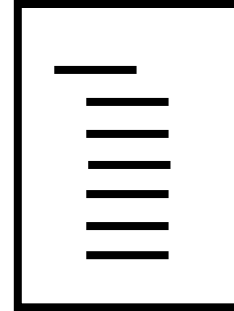
“Find the intersection of two sorted lists”



$O(m \cdot n)$

What we have now

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



$O(m+n)$

<https://bitbucket.org/tjknoth/resyn>