Type Assisted Synthesis of Programs with Algebraic Data Types

Jeevana Priya Inala

MIT

Collaborators: Xiaokang Qiu (MIT), Ben Lerner (Brown), Armando Solar-Lezama (MIT)

Example - Desugaring a simple language

ADT Definitions

```
adt srcAST {
  NumS { int v; }
  TrueS { }
  FalseS { }
  BinaryS { opcode op; srcAST a; srcAST b; }
  BetweenS{ srcAST a; srcAST b; srcAST c; }
}
```

a < b < c

```
adt dstAST {
  NumD { int v; }
  BoolD { bit v; }
  BinaryD { opcode op; dstAST a; dstAST b; }
}
```

Example - Desugaring a simple language

ADT Definitions

```
adt srcAST {
  NumS { int v; }
  TrueS { }
  FalseS { }
  BinaryS { opcode op; srcAST a; srcAST b; }
  BetweenS{ srcAST a; srcAST b; srcAST c; }
}
```

```
adt dstAST {
  NumD { int v; }
  BoolD { bit v; }
  BinaryD { opcode op; dstAST a; dstAST b; }
}
```

Specification

interpretSrc(s) == interpretDst(desugar(s))

Example - Desugaring a simple language

Function to be synthesized

```
dstAST desugar(srcAST s) {




}}}
```

Example - Desugaring a simple language

Function to be synthesized

```
dstAST desugar(srcAST s) {
  if (s == null) return null;
  switch(s){
  repeat_case: {
    dstAST a = desugar(s.??);
    dstAST b = desugar(s.??);
    dstAST c = desugar(s.??);
    return ??(3, {a, b, c, s.??});

}}}
```

repeat_case
General structure of pattern matching

Example - Desugaring a simple language

Function to be synthesized

```
dstAST desugar(srcAST s) {
  if (s == null) return null;
  switch(s){
  repeat_case: {
    dstAST a = desugar(s.??);
    dstAST b = desugar(s.??);
    dstAST c = desugar(s.??);
    return ??(3, {a, b, c, s.??});

}}}
```

Field selector hole
Choice among fields of s

Example - Desugaring a simple language

Function to be synthesized

```
dstAST desugar(srcAST s) {
 if (s == null) return null;
 switch(s){
 repeat_case: {
   dstAST a = desugar(s.??);
   dstAST b = desugar(s.??);
   dstAST c = desugar(s.??);
   return ??(3, {a, b, c, s.??});

}}}
```

Generalized constructor
Construct arbitrary ADT tree of depth at most 3

Example - Desugaring a simple language

Function to be synthesized

```
dstAST desugar(srcAST s) {
  if (s == null) return null;
  switch(s){
  repeat_case: {
    dstAST a = desugar(s.??);
    dstAST b = desugar(s.??);
    dstAST c = desugar(s.??);
    return ??(3, {a, b, c, s.??});

}}}
```

LOC: 7

No. of possible functions from template ~ 2^{110}

Example - Desugaring a simple language

Output

```
dstAST desugar(srcAST s) {
  if (s == null) return null;
  switch(s){
  case NumS: { return new NumD(v = s.v); }
  .....
  case BetweenS: {
            dstAST a = desugar(s.a);
            dstAST b = desugar(s.b);
            dstAST c = desugar(s.c);
        return new BinaryD(op = new AndOp(),
            a = new BinaryD(op = new LtOp(),
                          a = a, b = b),
            b = new BinaryD(op = new LtOp(),
                          a = b, b = c)); }
  .....
}}
```

LOC: 22

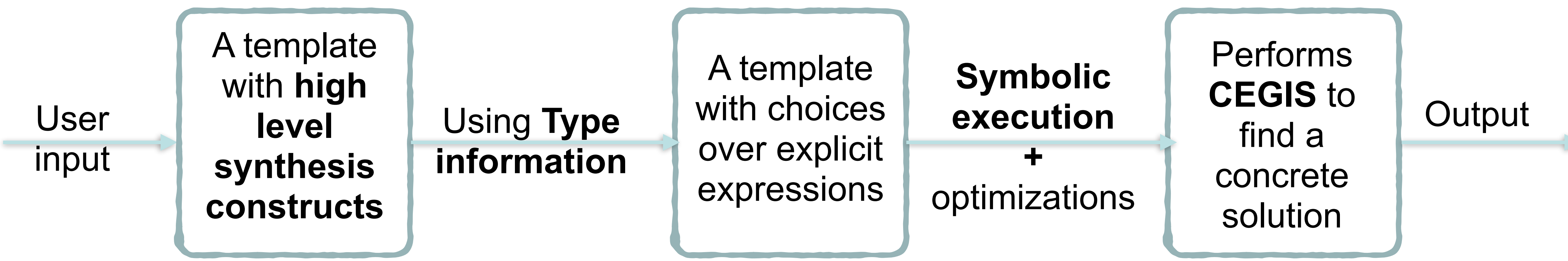
Synthesis Time: 36s

Technical Challenges

- Huge search space
  - On the order of 2^{100} - 2^{500}
- Complex specifications
  - Like interpreters
- High degree of recursion

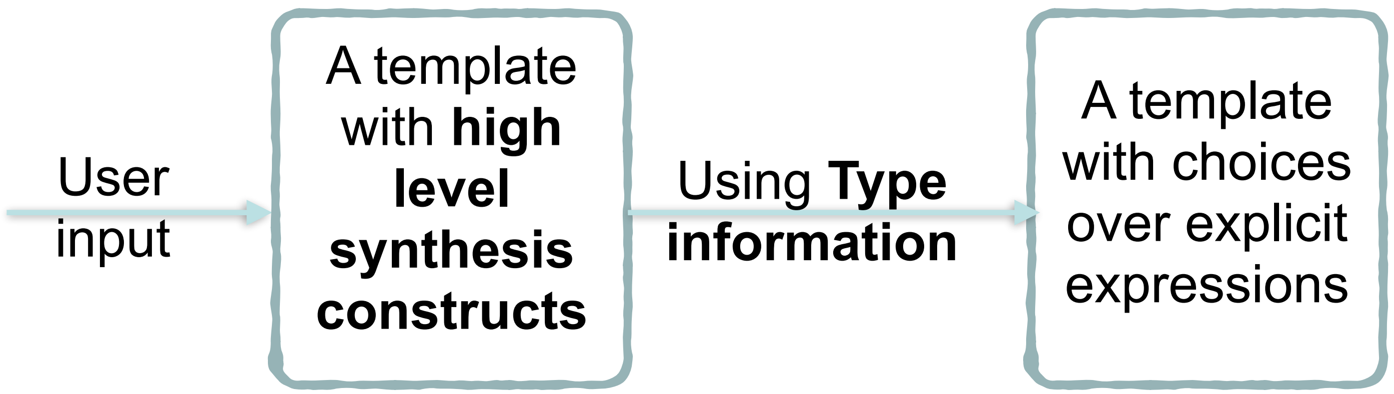
Approach

- Constraint based synthesis using Sketch [1]



1. A. Solar-Lezama. Program Synthesis By Sketching. PhD thesis, EECS Dept., UC Berkeley, 2008.

Type Directed Transformation



Type Directed Transformation

Requires propagating type information both top-down and bottom-up

BinaryS { opcode op; srcAST a; srcAST b; }

s.??

Type Directed Transformation

Requires propagating type information both top-down and bottom-up

BinaryS { opcode op; srcAST a; srcAST b; }

Type Directed Transformation

- Bi-directional rules of Pierce and Turner, 2000

$$\frac{T = \{\tau_0 \ldots \tau_i\}}{\Gamma \vdash e \xrightarrow{T} \{e_0 \ldots e_k\}}$$

General transformation rule

Type Directed Transformation

$$\frac{T^{'} = \{\tau \mid \ \tau\ has\ a\ field\ l : \tau_i\ and\ \tau_i \in T\ \} \quad \Gamma \vdash e \xrightarrow{T'} \{e_0 \ldots e_k\}}{\Gamma \vdash e.?? \ \xrightarrow{T} \ \{e_i.l_j \mid \ e_i.l_j : \tau \ \ j \in [0,k] \ \ and\ \tau \in T\}}$$

Type Directed Transformation

$$\frac{T^{'} = \{\tau \mid \ \tau \ has \ a \ field \ l : \tau_i \ and \ \tau_i \in T \ \} \qquad \Gamma \vdash e \xrightarrow{T'} \ \{e_0 \ldots e_k\}}{\Gamma \vdash e.?? \ \xrightarrow{T} \ \{e_i.l_j \mid \ e_i.l_j : \tau \ \ j \in [0,k] \ \ and \ \tau \in T\}}$$

s.**??** $\xrightarrow{\{srcAST\}}$

Type Directed Transformation

$$T' = \{\tau \mid \tau\ has\ a\ field\ l : \tau_i\ and\ \tau_i \in T\ \}$$

$$\frac{\Gamma \vdash e \xrightarrow{T'} \{e_0 \ldots e_k\}}{\Gamma \vdash e.?? \xrightarrow{T} \{e_i.l_j \mid e_i.l_j : \tau \quad j \in [0,k]\ \ and\ \tau \in T\}}$$

T' = {BinaryS, BetweenS}

s.?? $\xrightarrow{\{srcAST\}}$

Type Directed Transformation

$$\frac{T^{'} = \{\tau \mid \tau\ has\ a\ field\ l : \tau_i\ and\ \tau_i \in T\ \} \quad \Gamma \vdash e \xrightarrow{T^{'}} \{e_0 \ldots e_k\}}{\Gamma \vdash e.?? \xrightarrow{T} \{e_i.l_j \mid e_i.l_j : \tau \quad j \in [0, k]\ and\ \tau \in T\}}$$

T' = {BinaryS, BetweenS}

s $\xrightarrow{\text{\{BinaryS, BetweenS\}}}$ {s}

s.?? $\xrightarrow{\text{\{srcAST\}}}$

Type Directed Transformation

$$\frac{T^{'} = \{\tau \mid \tau\ has\ a\ field\ l : \tau_i\ and\ \tau_i \in T\ \} \qquad \Gamma \vdash e \xrightarrow{T'} \{e_0 \dots e_k\}}{\Gamma \vdash e.?? \ \xrightarrow{T} \{e_i.l_j \mid e_i.l_j : \tau \ \ j \in [0,k]\ \ and\ \tau \in T\}}$$

T' = {BinaryS, BetweenS}

$$s \xrightarrow{\{BinaryS,\ BetweenS\}} \{s\}$$

$$s.?? \xrightarrow{\{srcAST\}} \{s.a,\ s.b\}$$

Type Directed Transformation

$$\frac{\begin{array}{c} T^{'} = \{\tau \mid \tau\ has\ a\ field\ l : \tau_i\ and\ \tau_i \in T\ \} \\ \Gamma \vdash e \xrightarrow{T'} \{e_0 \ldots e_k\} \end{array}}{\Gamma \vdash e.?? \xrightarrow{T} \{e_i.l_j \mid e_i.l_j : \tau \quad j \in [0,k]\ \ and\ \tau \in T\}}$$

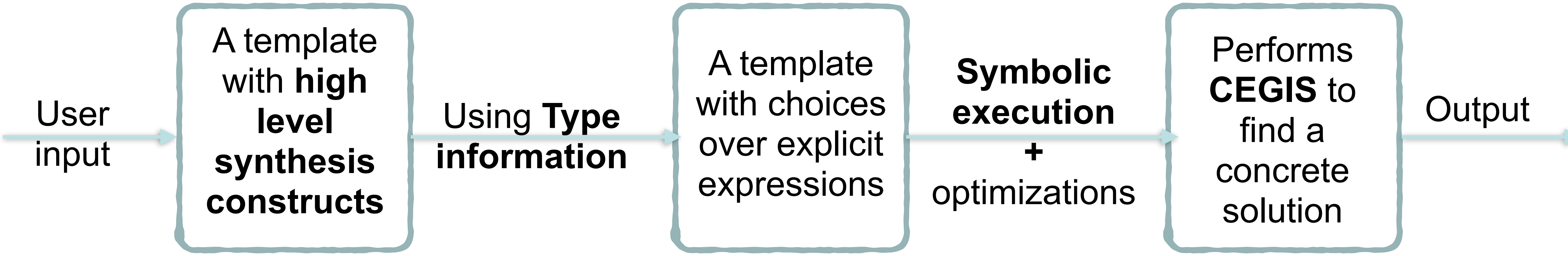
T' = {BinaryS, BetweenS}

s $\xrightarrow{\text{\{BinaryS, BetweenS\}}}$ {s}

s.?? $\xrightarrow{\text{\{srcAST\}}}$ {s.a, s.b}

s.??.??

Synthesis


User input
A template with **high level synthesis constructs**
Using **Type information**
A template with choices over explicit expressions
Symbolic execution
+
optimizations
Performs **CEGIS** to find a concrete solution
Output

Synthesis

- Inlines function calls and unrolls loops and creates a formula to encode to the SAT solver
- Uses Counter Example Guided Inductive Synthesis (CEGIS)
- Optimizations to improve scalability

Optimizations

1. Merging recursive calls with mutually exclusive path conditions

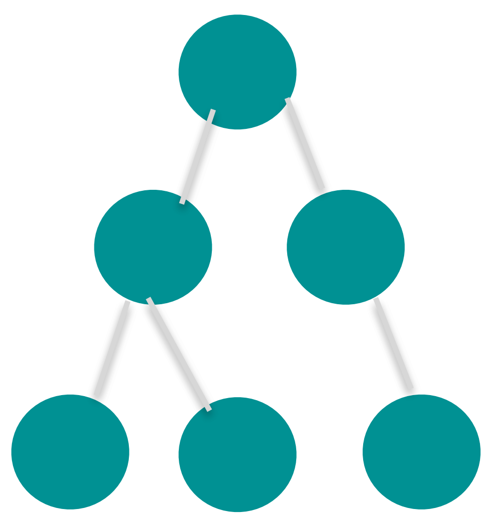
```
if (w)
    x = foo(a, b);
else
    y = foo(c, d);
```

→

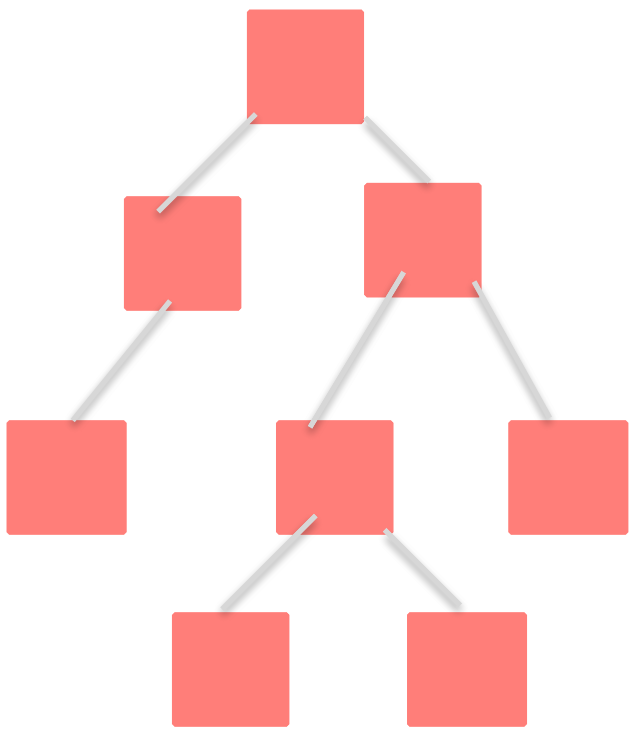
```
t = foo(w?a:c, w?b:d);
if (w)
    x = t;
else
    y = t;
```

Optimizations

2. Use specification as an invariant to abstract recursive calls



Input AST node

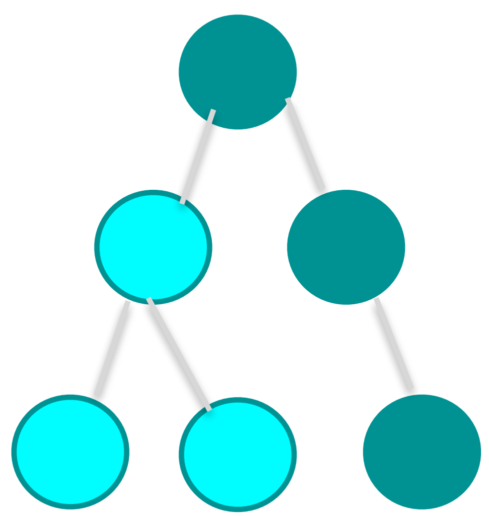


Desugared AST node

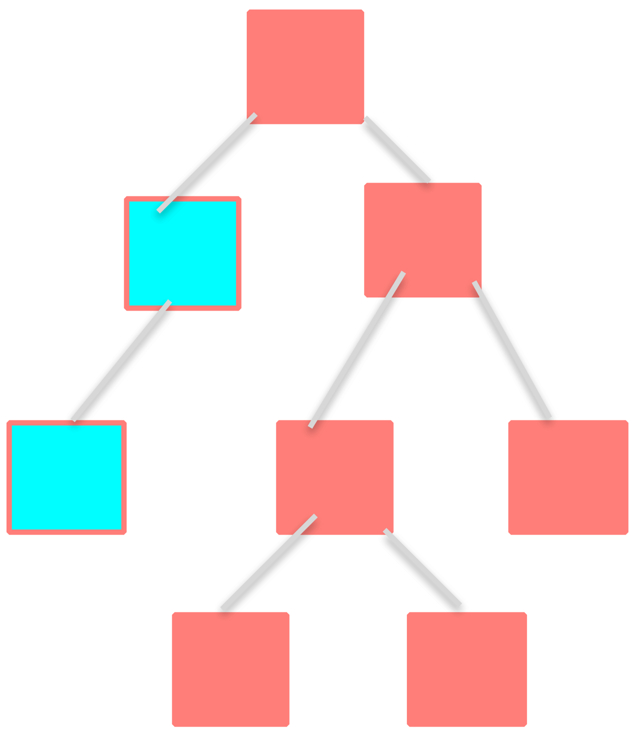
Specification: interpretSrc(s) == interpretDst(desugar(s))

Optimizations

2. Use specification as an invariant to abstract recursive calls



Input AST node

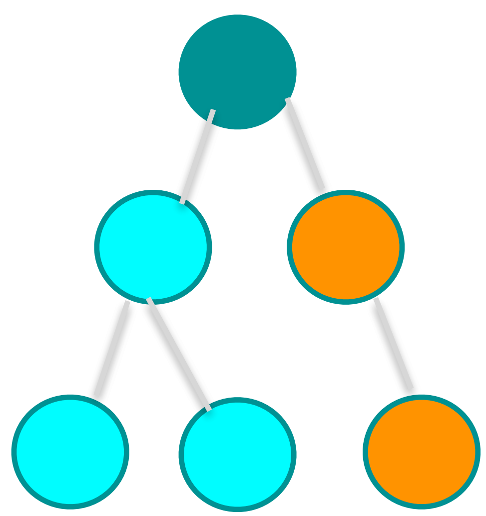


Desugared AST node

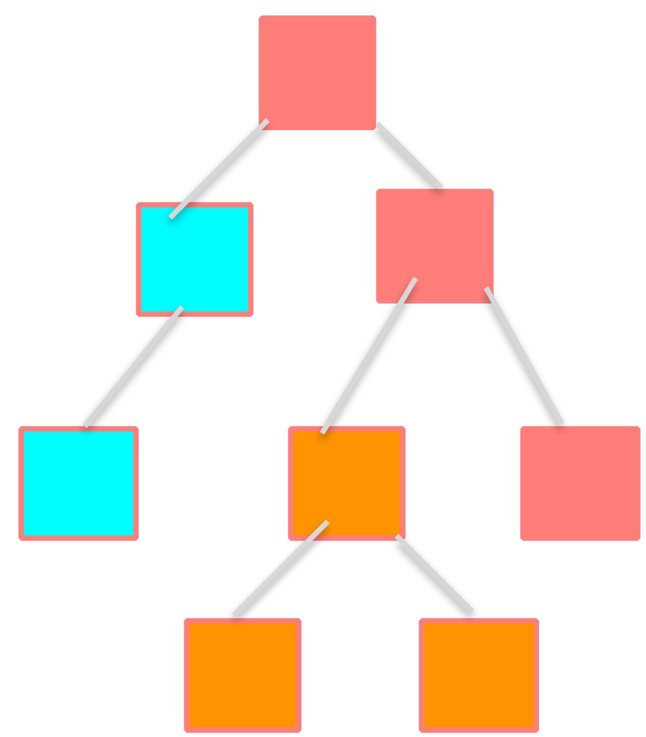
Specification: interpretSrc(s) == interpretDst(desugar(s))

Optimizations

2. Use specification as an invariant to abstract recursive calls



Input AST node

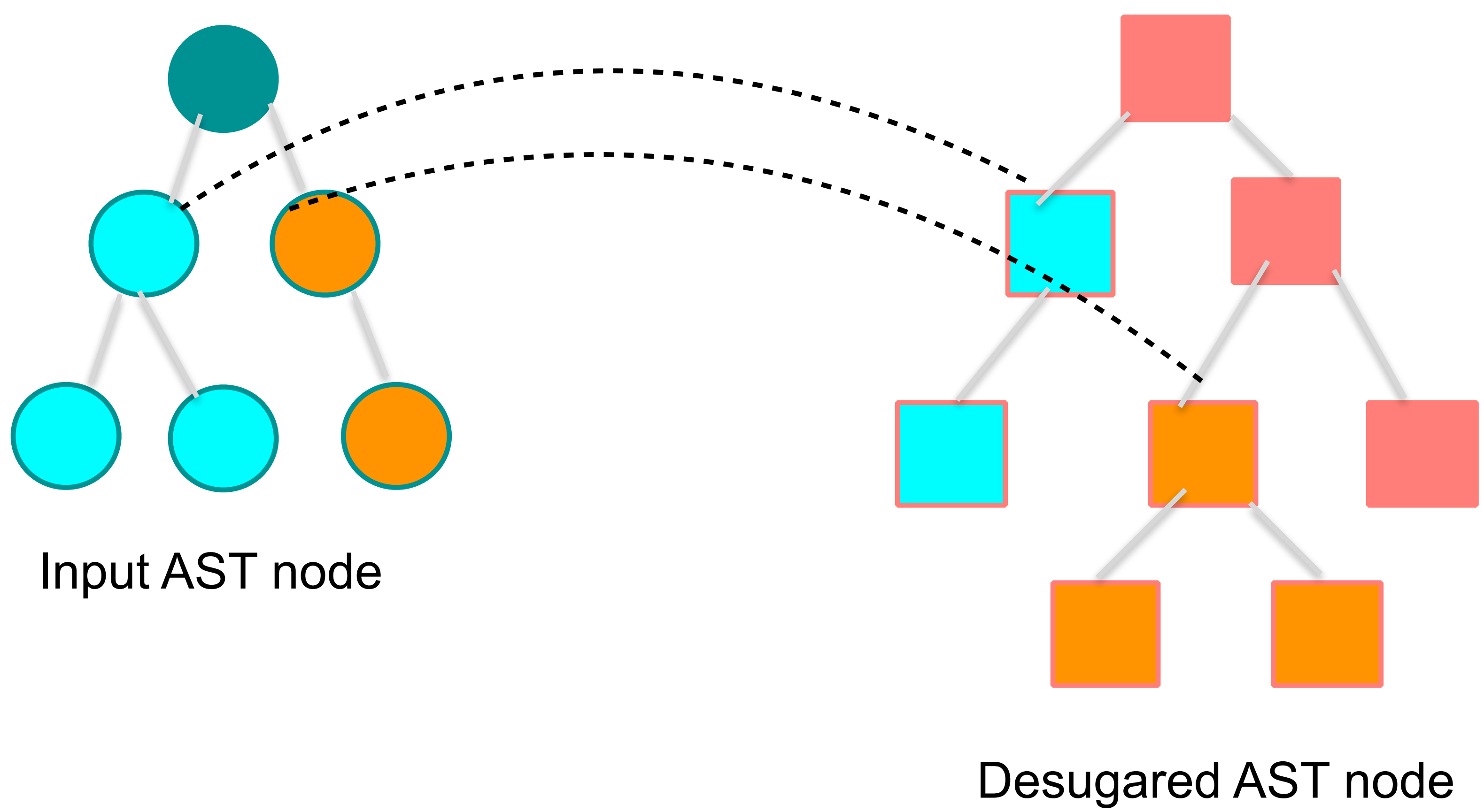


Desugared AST node

Specification: interpretSrc(s) == interpretDst(desugar(s))

Optimizations

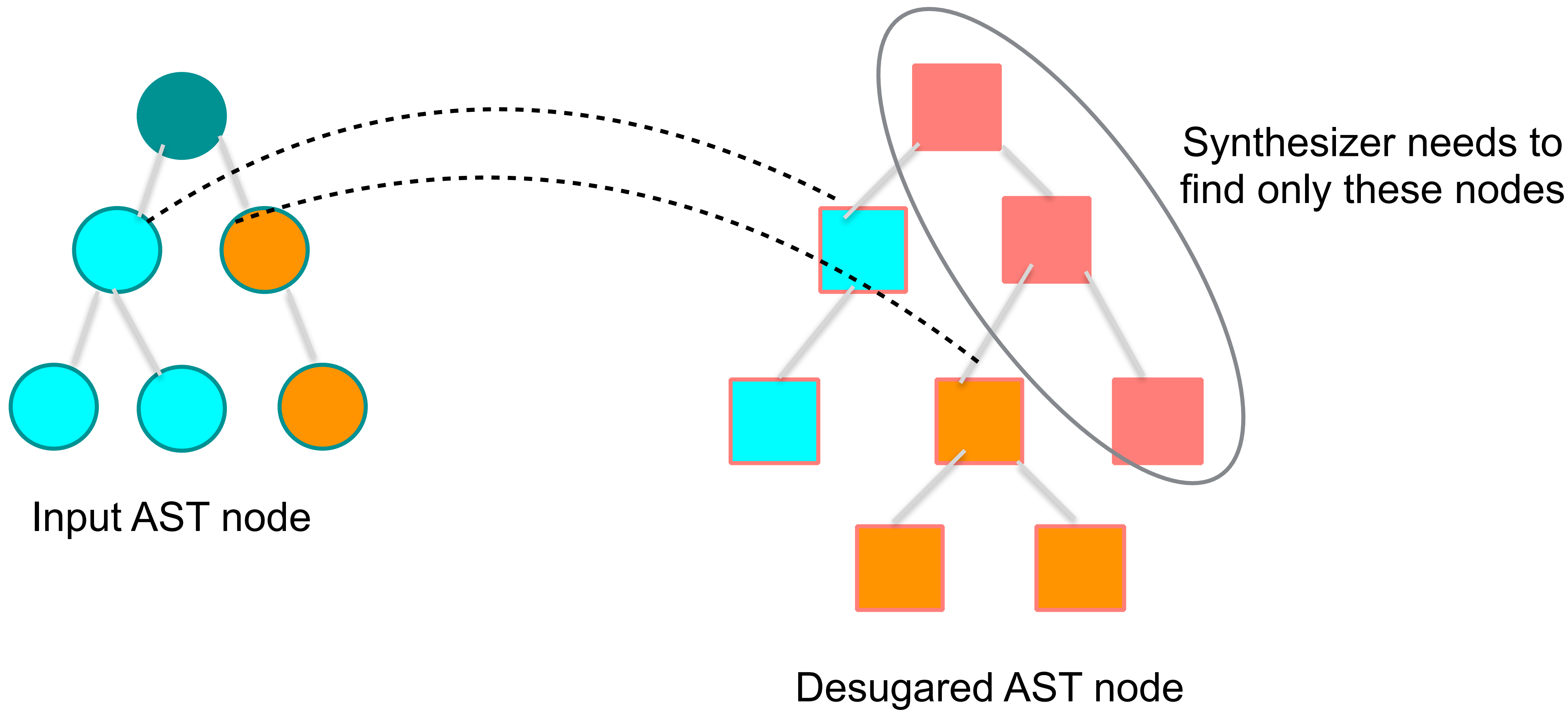
2. Use specification as an invariant to abstract recursive calls



Specification: interpretSrc(s) == interpretDst(desugar(s))

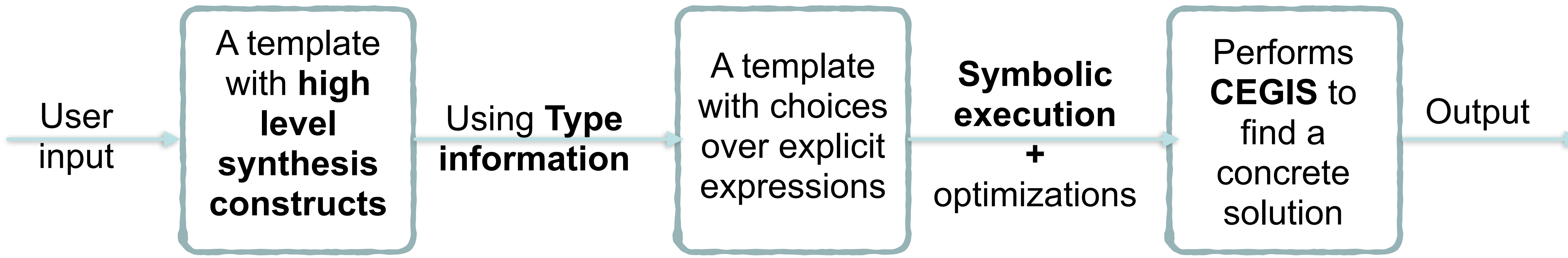
Optimizations

2. Use specification as an invariant to abstract recursive calls



Specification: interpretSrc(s) == interpretDst(desugar(s))

Synthesis



User input
A template with **high level synthesis constructs**
Using **Type information**
A template with choices over explicit expressions
Symbolic execution + optimizations
Performs **CEGIS** to find a concrete solution
Output

Evaluation

Benchmark	Runtime (s)	# of program choices
Insertion into a binary tree	18.42	2^72
Simple language desugaring	36.36	2^110
Simple language desugaring with state	577.19	2^141
Booleans to Lambda Calculus	114.14	2^541
Pairs to Lambda Calculus	683.55	2^183
AST optimatizations	163.09	2^162
Type constraints for Lambda Calculus	496.12	2^149

Conclusion

- A system to synthesize functions on Algebraic Data Types from high level templates
- Uses a combination of type inference and symbolic solving
- Can synthesize complex functions like desugaring

THANK YOU