Optimizing Synthesis with Metasketches

James Bornholt
Emina Torlak
Dan Grossman
Luis Ceze
University of Washington
Program synthesis
Specification \rightarrow \text{Program synthesis}
Program specification

Specified

Program synthesis

Generated
Program synthesis

Specification

\[ f(x) = 4x \]

Program
Program synthesis

Specification
\[ f(x) = 4x \]

Program
\[ x+x+x+x \]
**Program Specification**

\[ f(x) = 4x \]

**Program synthesis**

\[ x+x+x+x \]

**Compilation**

[PLDI’14]

**Data Structures**

[PLDI’15]

**End-user Programming**

[POPL’11]

**Executable Biology**

[POPL’13]

**Browser Layout**

[PPoPP’13]

**Cache Protocols**

[PLDI’13]
Often looking for an *optimal* solution, not just any correct program.
Program synthesis

Often looking for an *optimal* solution, not just any correct program.

There are many programs, so tools must control search strategy.

Specification

Compilation

Data Structures

End-user Programming

Executable Biology

Browser Layout

Cache Protocols
Often looking for an optimal solution, not just any correct program.

There are many programs, so tools must control search strategy.

Compilation [PLDI’14]

Data Structures [PLDI’15]

End-user Programming [POPL’11]

Executable Biology [POPL’13]

Browser Layout [PPoPP’13]

Cache Protocols [PLDI’13]
Program specification → Program synthesis → Program
Metasketches

Specification

Program synthesis

Program
Metasketches

A framework that makes search strategy and optimality part of the problem definition

Specification → Program synthesis → Program
Metasketches
Design and structure
Metasketches
Design and structure

Synapse
A metasketch solver
Metasketches
Design and structure

Synapse
A metasketch solver

Results
Better solutions, faster
Background
Syntax-guided synthesis

Metasketches
Design and structure

Synapse
A metasketch solver

Results
Better solutions, faster
Syntax-guided synthesis

Specification → Program synthesis → Program
Syntax-guided synthesis

Specification → Program synthesis → Program

Sketch
Syntax-guided synthesis

Specification → Program synthesis → Program

Sketch

def f(x):
    return Expr

Expr := x | ?? | Expr op Expr
op := + | * | - | >> | <<
?? := integer constant
Syntax-guided synthesis: guess, check, learn

```python
def f(x):
    return Expr

Expr := x | ?? | Expr op Expr
op := + | * | - | >> | <<
?? := integer constant
```
def f(x):
    return Expr

Expr := x | ?? | Expr op Expr
op := + | * | - | >> | <<
?? := integer constant
Syntax-guided synthesis: guess, check, learn

Counterexample-guided inductive synthesis [Solar-Lezama et al, 2006]
Syntax-guided synthesis: guess, check, learn

Counterexample-guided inductive synthesis [Solar-Lezama et al, 2006]

Semantics

Syntax

\[ f(x) = 4x \]
Syntax-guided synthesis: guess, check, learn

Counterexample-guided inductive synthesis [Solar-Lezama et al, 2006]
Syntax-guided synthesis: guess, check, learn
Counterexample-guided inductive synthesis [Solar-Lezama et al, 2006]
Syntax-guided synthesis: guess, check, learn

Counterexample-guided inductive synthesis [Solar-Lezama et al, 2006]

Semantics

Syntax

\[ f(x) = 4x \]
Syntax-guided synthesis: guess, check, learn

Counterexample-guided inductive synthesis [Solar-Lezama et al, 2006]

Semantics

Syntax

\[ f(x) = 4x \]
Syntax-guided synthesis: guess, check, learn

Counterexample-guided inductive synthesis [Solar-Lezama et al, 2006]

Semantics

Syntax

\[ f(x) = 4x \]

\[ x+x+x+x \]
Syntax-guided synthesis: guess, check, learn

Counterexample-guided inductive synthesis [Solar-Lezama et al, 2006]

1. **Search order** is critical
Syntax-guided synthesis: guess, check, learn
Counterexample-guided inductive synthesis [Solar-Lezama et al, 2006]

1. **Search order** is critical
2. Desire **optimal** solutions
Syntax-guided synthesis: guess, check, learn
Counterexample-guided inductive synthesis [Solar-Lezama et al, 2006]

1. **Search order** is critical
2. Desire **optimal** solutions
Background
Syntax-guided synthesis

Metasketches
Design and structure

Synapse
A metasketch solver

Results
Better solutions, faster
**Background**
Syntax-guided synthesis

**Metasketches**
Design and structure

**Synapse**
A metasketch solver

**Results**
Better solutions, faster
Metasketches express structure and strategy

1. **Search order** is critical
2. Desire **optimal** solutions
Metasketches express structure and strategy

1. **Search order** is critical
2. Desire **optimal** solutions

A metasketch contains:

1. structured candidate space \((\mathcal{S}, \preceq)\)
Metasketches express structure and strategy

1. **Search order** is critical
2. Desire **optimal** solutions

**A metasketch contains:**

1. structured candidate space \((\mathcal{S}, \preceq)\)
2. cost function \((\kappa)\)
Metasketches express structure and strategy

1. **Search order** is critical
2. Desire **optimal** solutions

A metasketch contains:

1. structured candidate space \((\mathcal{S}, \preceq)\)
2. cost function \((\kappa)\)
3. gradient function \((g)\)
Metasketches express structure and strategy

1. **Search order** is critical
2. Desire **optimal** solutions

A metasketch contains:

1. structured candidate space \( (\mathcal{S}, \preceq) \)
2. cost function \( (\kappa) \)
3. gradient function \( (g) \)
Metasketches express structure and strategy

1. **Search order** is critical
2. Desire **optimal** solutions

A metasketch contains:

1. structured candidate space \((\mathcal{S}, \preceq)\)
2. cost function \((\kappa)\)
3. gradient function \((g)\)
Metasketches express structure and strategy

1. **Search order** is critical
2. Desire **optimal** solutions

A metasketch contains:

1. structured candidate space \((\mathcal{S}, \preceq)\)
2. cost function \((\kappa)\)
3. gradient function \((g)\)
Metasketches express structure and strategy

1. **Search order** is critical
2. Desire **optimal** solutions

A metasketch contains:

1. structured candidate space ($S, \preceq$)
2. cost function ($\kappa$)
3. gradient function ($g$)
Metasketches express structure and strategy

1. **Search order** is critical
2. Desire **optimal** solutions

A metasketch contains:

1. structured candidate space \((\mathcal{S}, \leq)\)
2. cost function \((\kappa)\)
3. gradient function \((g)\)
Metasketches express structure and strategy

1. structured candidate space \((\mathcal{S}, \preceq)\)

2. cost function \((\kappa)\)

3. gradient function \((g)\)
Metasketches express structure and strategy

1. structured candidate space ($\mathcal{S}$, $\preceq$)

A fragmentation of the candidate space, and an ordering on those fragments.

2. cost function ($\kappa$)

3. gradient function ($g$)
Metasketches express structure and strategy

1. **structured candidate space** ($\mathcal{S}$, $\preceq$)

   A fragmentation of the candidate space, and an ordering on those fragments.

2. **cost function** ($\kappa$)

3. **gradient function** ($g$)

$\mathcal{S} = \text{set of all SSA programs}$
Metasketches express structure and strategy

1. structured candidate space \((\mathcal{S}, \preceq)\)

   A fragmentation of the candidate space, and an ordering on those fragments.

\[ \mathcal{S} = \text{set of all SSA programs} \]

2. cost function \((\kappa)\)

3. gradient function \((g)\)
Metasketches express structure and strategy

1. structured candidate space \((\mathcal{S}, \preceq)\)
   - a countable set \(\mathcal{S}\) of sketches
   - a total order \(\preceq\) on \(\mathcal{S}\)

\(\mathcal{S} = \) set of all SSA programs

A fragmentation of the candidate space, and an ordering on those fragments.

2. cost function \((\kappa)\)

3. gradient function \((g)\)
Metasketches express structure and strategy

1. structured candidate space \((\mathcal{S}, \preceq)\)
   - a countable set \(\mathcal{S}\) of sketches
   - a total order \(\preceq\) on \(\mathcal{S}\)

\[ S = \text{set of all SSA programs} \]

\[ S_3 \quad \text{(SSA programs of length 3)} \]

\[
\begin{align*}
\text{def } f(x): \\
r_1 &= \text{???}_{\text{op}}(\text{??}\{x\}) \\
r_2 &= \text{???}_{\text{op}}(\text{??}\{x, r_1\}) \\
r_3 &= \text{???}_{\text{op}}(\text{??}\{x, r_1, r_2\}) \\
\text{return } r_3 
\end{align*}
\]

2. cost function \((\kappa)\)

3. gradient function \((g)\)
Metasketches express structure and strategy

1. structured candidate space \((\mathcal{S}, \preceq)\)
   - a countable set \(\mathcal{S}\) of sketches
   - a total order \(\preceq\) on \(\mathcal{S}\)

2. cost function \((\kappa)\)

3. gradient function \((g)\)
Metasketches express structure and strategy

1. **structured candidate space** ($\mathcal{S}$, $\preceq$)
   - a countable set $\mathcal{S}$ of sketches
   - a total order $\preceq$ on $\mathcal{S}$

A fragmentation of the candidate space, and an ordering on those fragments.

$$\mathcal{S} = \text{set of all SSA programs}$$

**$\mathcal{S}_3$** (SSA programs of length 3)

```python
def f(x):
    r_1 = ??_op(??_{x})
    r_2 = ??_op(??_{x, r_1})
    r_3 = ??_op(??_{x, r_1, r_2})
    return r_3
```

2. cost function ($\kappa$)
3. gradient function ($g$)
Metasketches express structure and strategy

1. structured candidate space ($\mathcal{S}$, $\preceq$)
   - a countable set $\mathcal{S}$ of sketches
   - a total order $\preceq$ on $\mathcal{S}$

   A fragmentation of the candidate space, and an ordering on those fragments.

   $\mathcal{S}$ = set of all SSA programs

2. cost function ($\kappa$)

3. gradient function ($g$)
Metasketches express structure and strategy

1. structured candidate space \((\mathcal{S}, \preceq)\)
   - a countable set \(\mathcal{S}\) of sketches
   - a total order \(\preceq\) on \(\mathcal{S}\)

\[\mathcal{S} = \text{set of all SSA programs}\]

A fragmentation of the candidate space, and an ordering on those fragments.

2. cost function \((\kappa)\)

3. gradient function \((g)\)

\[
\begin{align*}
\text{def } f(x): \\
& r_1 = \text{??}_\text{op}(\text{??}\{x\}) \\
& r_2 = \text{??}_\text{op}(\text{??}\{x, r_1\}) \\
& r_3 = \text{??}_\text{op}(\text{??}\{x, r_1, r_2\}) \\
& \text{return } r_3
\end{align*}
\]
Metasketches express structure and strategy

1. structured candidate space ($\mathcal{S}$, $\preceq$)
   - a countable set $\mathcal{S}$ of sketches
   - a total order $\preceq$ on $\mathcal{S}$

   A fragmentation of the candidate space, and an ordering on those fragments.

   $\mathcal{S}$ = set of all SSA programs

   $\mathcal{S}_3$ (SSA programs of length 3)

   ```python
   def f(x):
       r_1 = ???_{??op}(??_{?}{x})
       r_2 = ???_{??op}(??_{?}{x, r_1})
       r_3 = ???_{??op}(??_{?}{x, r_1, r_2})
       return r_3
   ```

2. cost function ($\kappa$)

3. gradient function ($g$)

Ordering expresses high-level search strategy.
Here, $\preceq$ expresses iterative deepening.
Metasketches express structure and strategy

1. **structured candidate space** \((\mathcal{S}, \preceq)\)
   - a countable set \(\mathcal{S}\) of sketches
   - a total order \(\preceq\) on \(\mathcal{S}\)

2. **cost function** \((\kappa)\)

3. **gradient function** \((g)\)

\(\mathcal{S}\) = set of all SSA programs

A fragmentation of the candidate space, and an ordering on those fragments.

Implemented as a generator that returns the next sketch in the space
Metasketches express structure and strategy

1. structured candidate space \((\mathcal{S}, \preceq)\)
   - a countable set \(\mathcal{S}\) of sketches
   - a total order \(\preceq\) on \(\mathcal{S}\)

2. cost function \((\kappa)\)

3. gradient function \((g)\)

\[
\mathcal{S} = \text{set of all SSA programs}
\]

Semantics
Metasketches express structure and strategy

1. structured candidate space \((\mathcal{S}, \preceq)\)
   - a countable set \(\mathcal{S}\) of sketches
   - a total order \(\preceq\) on \(\mathcal{S}\)

2. cost function \((\kappa)\)

3. gradient function \((g)\)

\[\mathcal{S} = \text{set of all SSA programs}\]
Metasketches express structure and strategy

1. structured candidate space \((\mathcal{S}, \leq)\)
   - a countable set \(\mathcal{S}\) of sketches
   - a total order \(\leq\) on \(\mathcal{S}\)

2. cost function \((\kappa)\)

3. gradient function \((g)\)

\(\mathcal{S}\) = set of all SSA programs
Metasketches express structure and strategy

1. structured candidate space ($\mathcal{S}, \preceq$)
   - a countable set $\mathcal{S}$ of sketches
   - a total order $\preceq$ on $\mathcal{S}$

2. cost function ($\kappa$)

3. gradient function ($g$)

$\mathcal{S} = \text{set of all SSA programs}$

Semantic redundancy in the search space.

Semantics
Metasketches express structure and strategy

1. structured candidate space \((\mathcal{S}, \preceq)\)
   - a countable set \(\mathcal{S}\) of sketches
   - a total order \(\preceq\) on \(\mathcal{S}\)

2. cost function \((\kappa)\)

3. gradient function \((g)\)

\[ \mathcal{S} = \text{set of all SSA programs} \]

Semantic redundancy in the search space.

Structure constraints eliminate some overlap between sketches.
Metasketches express structure and strategy

1. structured candidate space \((\mathcal{S}, \preceq)\)
   - a countable set \(\mathcal{S}\) of sketches
   - a total order \(\preceq\) on \(\mathcal{S}\)

\[\mathcal{S} = \text{set of all SSA programs}\]

\[\mathcal{S}_3\] (SSA programs of length 3)

2. cost function \((\kappa)\)

3. gradient function \((g)\)

\[\text{def } f(x): \]
\[r_1 = ??_\text{op}(??\{x\})\]
\[r_2 = ??_\text{op}(??\{x, r_1\})\]
\[r_3 = ??_\text{op}(??\{x, r_1, r_2\})\]
\[\text{return } r_3\]

Semantic redundancy in the search space.

Structure constraints eliminate some overlap between sketches.
Metasketches express structure and strategy

1. structured candidate space \((\mathcal{S}, \preceq)\)
   - a countable set \(\mathcal{S}\) of sketches
   - a total order \(\preceq\) on \(\mathcal{S}\)

2. cost function \((\kappa)\)

3. gradient function \((g)\)

\[ \mathcal{S} = \text{set of all SSA programs} \]

Semantics

\[ \langle S_2 \rangle \]

[S2]

[S3]

Structure constraints eliminate some overlap between sketches

Semantic redundancy in the search space.

Eliminate dead-code redundancy: assert that each \(r_1\) is read

```
def f(x):
    r_1 = ??_op(??_{x})
    r_2 = ??_op(??_{x, r_1})
    r_3 = ??_op(??_{x, r_1, r_2})
    return r_3
```
Cost functions rank candidate programs

1. structured candidate space \((\mathcal{S}, \preceq)\)

2. cost function \((\kappa)\)

\[
\kappa : \mathcal{L} \rightarrow \mathbb{R}
\]

assigns a numeric cost to each program in the language \(\mathcal{L}\)

3. gradient function \((g)\)

\[\mathcal{S} = \text{set of all SSA programs}\]
Cost functions rank candidate programs

1. structured candidate space \((\mathcal{S}, \preceq)\)

2. cost function \((\kappa)\)
   
   \(\kappa : \mathcal{L} \rightarrow \mathbb{R}\)
   
   assigns a numeric cost to each program in the language \(\mathcal{L}\)

3. gradient function \((g)\)

\[\mathcal{S} = \text{set of all SSA programs}\]
Cost functions rank candidate programs

1. structured candidate space \((S, \preceq)\)

2. cost function \((\kappa)\)
   \[
   \kappa : \mathcal{L} \rightarrow \mathbb{R}
   \]
   assigns a numeric cost to each program in the language \(\mathcal{L}\)

3. gradient function \((g)\)

\[S = \text{set of all SSA programs}\]

Cost functions can be based on both syntax and semantics (dynamic behavior)

\[\kappa(P) = i \quad \text{for} \ P \in S_i \in S\]

The number of variables defined in \(P\)
Gradient functions provide cost structure

1. structured candidate space \((\mathcal{S}, \preceq)\)
2. cost function \((\kappa)\)
3. gradient function \((g)\)

\[
g : \mathbb{R} \rightarrow 2^{\mathcal{S}}
\]

\(g(c)\) is the set of sketches in \(\mathcal{S}\) that may contain a solution \(P\) with \(\kappa(P) < c\)

\[
\mathcal{S} = \text{set of all SSA programs}
\]

\[
\kappa(P) = i \quad \text{for} \ P \in S_i \in \mathcal{S}
\]
Gradient functions provide cost structure

1. structured candidate space ($\mathcal{S}$, $\preceq$)
2. cost function ($\kappa$)
3. gradient function ($g$)

$g : \mathbb{R} \to 2^\mathcal{S}$

$g(c)$ is the set of sketches in $\mathcal{S}$ that may contain a solution $P$ with $\kappa(P) < c$

The gradient function overapproximates the behavior of $\kappa$ on $\mathcal{S}$

$\mathcal{S} = \text{set of all SSA programs}$

$\kappa(P) = i$ for $P \in S_i \in \mathcal{S}$
Gradient functions provide cost structure

1. structured candidate space \((\mathcal{S}, \preceq)\)

2. cost function \((\kappa)\)

3. gradient function \((g)\)

\[
g : \mathbb{R} \rightarrow 2^\mathcal{S}
\]

\(g(c)\) is the set of sketches in \(\mathcal{S}\) that may contain a solution \(P\) with \(\kappa(P) < c\)

\[
\kappa(P) = i \quad \text{for } P \in S_i \in \mathcal{S}
\]

\[
g(c) = \{ S_i \in \mathcal{S} \mid i < c \}
\]
Gradient functions provide cost structure

1. structured candidate space \((\mathcal{S}, \preceq)\)
2. cost function \((\kappa)\)
3. gradient function \((g)\)

\[ g : \mathbb{R} \to 2^\mathcal{S} \]
\[ g(c) \text{ is the set of sketches in } \mathcal{S} \text{ that may contain a solution } P \text{ with } \kappa(P) < c \]

\[ \mathcal{S} = \text{set of all SSA programs} \]

\[ \kappa(P) = i \text{ for } P \in S_i \in \mathcal{S} \]
\[ g(c) = \{ S_i \in \mathcal{S} | i < c \} \]

The gradient function overapproximates the behavior of \(\kappa\) on \(\mathcal{S}\)
Gradient functions provide cost structure

1. structured candidate space \((\mathcal{S}, \preceq)\)
2. cost function \((\kappa)\)
3. gradient function \((g)\)

\[ g : \mathbb{R} \to 2^\mathcal{S} \]

\( g(c) \) is the set of sketches in \( \mathcal{S} \) that may contain a solution \( P \) with \( \kappa(P) < c \)

\[ \mathcal{S} = \text{set of all SSA programs} \]

\[ \kappa(P) = i \quad \text{for } P \in S_i \in \mathcal{S} \]

\[ g(c) = \{ S_i \in \mathcal{S} \mid i < c \} \]
Gradient functions provide cost structure

1. structured candidate space \((\mathcal{S}, \preceq)\)
2. cost function \((\kappa)\)
3. gradient function \((g)\)

\[
g : \mathbb{R} \to 2^\mathcal{S}
\]

\(g(c)\) is the set of sketches in \(\mathcal{S}\) that may contain a solution \(P\) with \(\kappa(P) < c\)

\(\mathcal{S}\) = set of all SSA programs

\(\kappa(P) = i\) for \(P \in S_i \in \mathcal{S}\)

\(g(c) = \{ S_i \in \mathcal{S} \mid i < c \} \)

The gradient function overapproximates the behavior of \(\kappa\) on \(\mathcal{S}\)

Always sound for \(g\) to return all of \(\mathcal{S}\) if a tighter bound is unavailable.

\(g(c)\) always being finite is sufficient (not necessary) to guarantee termination.
Metasketches
Design and structure

1. structured candidate space \((S, \preceq)\)
2. cost function \((\kappa)\)
3. gradient function \((g)\)

\[ S = \text{set of all SSA programs} \]

\[ \kappa(P) = i \quad \text{for } P \in S_i \in S \]

\[ g(c) = \{ S_i \in S \mid i < c \} \]
Background
Syntax-guided synthesis

Metasketches
Design and structure

Synapse
A metasketch solver

Results
Better solutions, faster
Background
Syntax-guided synthesis

Metasketches
Design and structure

Synapse
A metasketch solver

Results
Better solutions, faster
Solving with two cooperative searches

\[ \langle \mathcal{S}, \preceq, \kappa, g \rangle \]

Coordinates the search for an optimal solution, offloading work to parallel local searches.
Solving with two cooperative searches

\[ \langle S, \preceq, \kappa, g \rangle \]

Coordinates the search for an optimal solution, offloading work to parallel local searches

An incremental form of CEGIS that can accept new information from the global search
Solving with two cooperative searches

\[ \langle S, \leq, \kappa, g \rangle \]

- Global search
- Local search
- Local search
- Local search

Diagram:

\[ S_1, S_2, S_3, S_4, S_5, S_6, S_7 \]
Solving with two cooperative searches

\[ \langle S, \leq, \kappa, g \rangle \]
Solving with two cooperative searches

\[ \langle S, \leq, \kappa, g \rangle \]
Solving with two cooperative searches

\( \langle S, \leq, \kappa, g \rangle \)

Global search

Local search

Local search

Local search

\( S_1 \)

\( S_2 \)

\( S_3 \)
Solving with two cooperative searches

\( \langle S, \leq, \kappa, g \rangle \)

Global search

Local search

Local search

Local search

\( S_1 \)

\( S_3 \)

\( S_4 \)

\( S_6 \)

\( S_5 \)

\( S_7 \)
Solving with two cooperative searches

\[ \langle S, \leq, \kappa, g \rangle \]

Global search

Local search

Local search

Local search

\[ \text{SAT} (P) \]
Solving with two cooperative searches

\[ \langle S, \leq, k, g \rangle \]
Solving with two cooperative searches

\[ \langle \mathcal{S}, \leq, \kappa, g \rangle \]

Global search

\[ \kappa(P) \]

Local search

\[ \kappa(P) \]

Local search

\[ \kappa(P) \]
Solving with two cooperative searches

\[ \langle S, \preceq, \kappa, g \rangle \]
Solving with two cooperative searches

\[ \langle \mathcal{S}, \preceq, \kappa, g \rangle \]
Solving with two cooperative searches

\[ \langle S, \preceq, \kappa, g \rangle \]

- Global search
- Local search
- Local search

Prune local search spaces using \( \kappa(P) \)

Prune global search space using \( g(\kappa(P)) \)
Solving with two cooperative searches

\[ \langle S, \leq, \kappa, g \rangle \]

Global search

\[ \kappa(P) \]

Local search

Prune local search spaces using \( \kappa(P) \)

Prune global search space using \( g(\kappa(P)) \)
Solving with two cooperative searches

\[ \langle S, \leq, \kappa, g \rangle \]

Global search

- Continues until all search spaces exhausted, yielding an optimal solution.

- Prune local search spaces using \( \kappa(P) \)

- Prune global search space using \( g(\kappa(P)) \)

Local search

Prune local search spaces using \( \kappa(P) \)

S1

S4

S3

S5

S6
Synapse implementation

\[ \langle S, \preceq, \kappa, g \rangle \]

Implemented in Rosette, a solver-aided extension of Racket
Synapse implementation

\[ \langle \mathcal{S}, \preceq, \kappa, g \rangle \]

- Implemented in Rosette, a solver-aided extension of Racket
- Local CEGIS searches can share counterexamples
Synapse implementation

\[ \langle \mathcal{S}, \preceq, \kappa, g \rangle \]

Implemented in Rosette, a solver-aided extension of Racket

Local CEGIS searches can share counterexamples

Local searches can time out, which weakens optimality
**Background**
Syntax-guided synthesis

**Metasketches**
Design and structure

**Synapse**
A metasketch solver

**Results**
Better solutions, faster
Background
Syntax-guided synthesis

Metasketches
Design and structure

Synapse
A metasketch solver

Results
Better solutions, faster
Evaluation questions

Is Synapse a practical approach to solving different kinds of synthesis problems?
  Approximate computing, array programs
Evaluation questions

Is Synapse a practical approach to solving different kinds of synthesis problems?
   Approximate computing, array programs

Can Synapse reason about complex cost functions?
Evaluation questions

Is Synapse a practical approach to solving different kinds of synthesis problems?
   Approximate computing, array programs

Can Synapse reason about complex cost functions?

In the paper:
   • Parallel speedup
   • Optimizations (structure constraints, sharing)
   • More kinds of problems
   • More complex cost functions
Synapse solves previously-intractable problems

**Parrot** benchmarks from approximate computing [Esmaelizadeh et al., 2012]

Find the most efficient approximate program within an error bound
Synapse solves previously-intractable problems

**Parrot** benchmarks from approximate computing [Esmaelizadeh et al., 2012]

Find the most efficient approximate program within an error bound
Synapse solves previously-intractable problems

**Parrot** benchmarks from approximate computing [Esmaelizadeh et al., 2012]

Find the most efficient approximate program within an error bound

![Graph showing solving time (secs) for different benchmarks](image)
Synapse solves standard benchmarks optimally

**Array Search** benchmarks from the syntax-guided synthesis (SyGuS) competition [Alur et al., 2015]

arraysearch-$n$: find program that searches lists of length $n$
Synapse solves standard benchmarks optimally

Array Search benchmarks from the syntax-guided synthesis (SyGuS) competition [Alur et al., 2015]

arraysearch-$n$: find program that searches lists of length $n$
Synapse solves standard benchmarks optimally

Array Search benchmarks from the syntax-guided synthesis (SyGuS) competition [Alur et al., 2015]

arraysearch-\(n\): find program that searches lists of length \(n\)
Is this a cat?
Synapse reasons about complex costs
Synapse reasons about complex costs

\[ \kappa(P) = \sum_i |P(x_i) - y_i| \]

*Classification error executes the program for each point in the training set*
Metasketches
Design and structure

Synapse
A metasketch solver

Results
Better solutions, faster
Metasketches
Design and structure

Synapse
A metasketch solver

Results
Better solutions, faster

synapse.uwplse.org