Optimizing synthesis with metasketches

James Bornholt, Emina Torlak, Dan Grossman, and Luis Ceze

University of Washington
1. introduction
2. metaskeches
3. synapse
4. results
synthesis with sketches
synthesis with sketches
specification

synthesis with sketches
specification

synthesis with sketches

implementation
$f(x) = 4 \cdot x$

**synthesis** with sketches

**implementation**
f(x) = 4*x

def f(x):
    return x+x+x+x
$$f(x) = 4x$$

```
def f(x):
    return x+x+x+x
```

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

synthesis with sketches

def f(x):
    return x + x + x + x

def f(x):
    return Expr

Expr := x | ?? | Expr op Expr
op := + | * | - | >> | <<
?? := integer constant

Counterexample-Guided Inductive Synthesis (CEGIS)
Counterexample-Guided Inductive Synthesis (CEGIS)

$f(x) = 4\times x$

def $f(x)$:
  return $x+x+x+x$

def $f(x)$:
  return $x+x+x+x$

**synthesis with sketches**

$Expr := x \mid ?? \mid Expr \ op \ Expr$

$op := + \mid * \mid - \mid >> \mid <<$

$?? := integer \ constant$

guess, check, learn
\[ f(x) = 4 \times x \]

def \( f(x) \):
    return \( \text{Expr} \)

\( \text{Expr} := x \mid \text{??} \mid \text{Expr} \text{ op} \text{ Expr} \)

\( \text{op} := + \mid \ast \mid - \mid >> \mid << \)

\( \text{??} := \text{integer constant} \)

guess, check, learn

candidate programs
def f(x):
    return x + x + x + x

def f(x):
    return x + 1
synthesis with sketches

def f(x):
    return x+x+x+x

def f(x):
    return x+1

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

candidate programs

f(0) ≠ 0

f(x) = 4*x
\[ f(x) = 4 \cdot x \]

def \textit{f(x)}:
    \textbf{return} \textit{Expr}

\textit{Expr} := \textit{x} \mid \textit{??} \mid \textit{Expr} \textit{op} \textit{Expr}

\textit{op} := + \mid * \mid - \mid >> \mid <<

?? := integer constant

def \textit{f(x)}:
    \textbf{return} \textit{x+1}

\textbf{candidate programs}

\textbf{guess, check, learn}

\textbf{synthesis with sketches}

\textbf{f(0) \neq 0}
**synthesis with sketches**

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

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

**Specification**

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

**Implementation**

```python
def f(x):
    return x+x+x+x
```
building a practical synthesizer is hard ...

def f(x):
    return x+x+x+x

candidate programs: 

\[ f(x) = 4 \times x \]
building a practical synthesizer is hard ...

1. pick the right search strategy

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

Candidate programs: $f(x) = 4 \times x$
building a practical synthesizer is hard ...

1. pick the right search strategy
2. find the best correct program

\[ f(x) = 4 \times x \]
building a practical synthesizer is hard ...

Existing tools hardcode both the search strategy and cost metric (if any), so are difficult to reuse for new domains.

1. pick the right search strategy
2. find the best correct program

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

```python
def f(x):
    return x >> 2
```

candidate programs
building a practical synthesizer is hard...

Existing tools hardcode both the search strategy and cost metric (if any), so are difficult to reuse for new domains.

1. pick the right search strategy
2. find the best correct program

Metasketches are a new way to express synthesis problems, making the search strategy and the cost function explicit in the problem definition.

\[ f(x) = 4^*x \]

```python
def f(x):
    return x >> 2
```
anatomy of a metasketch

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

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

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

candidate programs
anatomy of a metasketch

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

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

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** \((g)\)
anatomy of a metasketch (I)

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** \((g)\)

A set of sketches \(\mathcal{S}\) can express candidate spaces that cannot be expressed with a single sketch or a CFG.
anatomy of a metasketch (1)

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** ($g$)

A set of sketches $\mathcal{S}$ can express candidate spaces that cannot be expressed with a single sketch or a CFG.
anatomy of a metasketch (I)

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 ($\nabla$)

A set of sketches $\mathcal{S}$ can express candidate spaces that cannot be expressed with a single sketch or a CFG.

The ordering $\preceq$ on sketches expresses a high-level search strategy.
anatomy of a metasketch (1)

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** \((g)\)

A set of sketches \(\mathcal{S}\) can express candidate spaces that cannot be expressed with a single sketch or a CFG.

The ordering \(\preceq\) on sketches expresses a high-level search strategy.
anatomy of a metasketch (1)

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** ($g$)

```python
def f(x):
    r1 = [x|??] [+|...] [x|??]
    return r1

def f(x):
    r1 = [x|??] [+|...] [x|??]
    r2 = [x|r1|??] [+|...] [x|r1|??]
    return r2

def f(x):
    r1 = [x|??] [+|...] [x|??]
    r2 = [x|r1|??] [+|...] [x|r1|??]
    r3 = [x|r1|r2|??] [+|...] [x|r1|r2|??]
    return r3
```

$\mathcal{S}$, $\preceq$ (SSA with iterative deepening)
anatomy of a metasketch (I)

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 ($g$)

```python
def f(x):
    r1 = [x|??] [+|...] [x|??]
    return r1

def f(x):
    r1 = [x|??] [+|...] [x|??]
    r2 = [x|r1|??] [+|...] [x|r1|??]
    return r2

def f(x):
    r1 = [x|??] [+|...] [x|??]
    r2 = [x|r1|??] [+|...] [x|r1|??]
    r3 = [x|r1|r2|??] [+|...] [x|r1|r2|??]
    return r3
```
anatomy of a metasketch (1)

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 ($g$)

```python
def f(x):
    r1 = [x|??] [+|...] [x|??]
    return r1

def f(x):
    r1 = e11 [+|...] e12
    r2 = e21 [+|...] e22
    return r2

def f(x):
    r1 = [x|??] [+|...] [x|??]
    r2 = [x|r1|??] [+|...] [x|r1|??]
    r3 = [x|r1|r2|??] [+|...] [x|r1|r2|??]
    return r3
```
anatomy of a metasketch (1)

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** $(g)$

\[
\begin{align*}
def f(x): & \\
    r1 &= [x|??] [+|...] [x|??] \\
    \text{return } r1
\end{align*}
\]

\[
\begin{align*}
def f(x): & \\
    r1 &= e11 [+|...] e12 \\
    r2 &= e21 [+|...] e22 \\
    \text{assert } r1 == e21 || r1 == e22 \\
    \text{return } r2
\end{align*}
\]

\[
\begin{align*}
def f(x): & \\
    r1 &= [x|??] [+|...] [x|??] \\
    r2 &= [x|r1|??] [+|...] [x|r1|??] \\
    r3 &= [x|r1|r2|??] [+|...] [x|r1|r2|??] \\
    \text{return } r3
\end{align*}
\]
anatomy of a metasketch (I)

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** \((g)\)

\(\mathcal{S}, \preceq\) (SSA with iterative deepening)

Structure constraints help reduce semantic redundancies in the search space.

```python
def f(x):
    r1 = [x|??] [+|...] [x|??]
    return r1

def f(x):
    r1 = e11 [+|...] e12
    r2 = e21 [+|...] e22
    assert r1 == e21 || r1 == e22
    return r2

def f(x):
    r1 = e11 [+|...] e12
    r2 = e21 [+|...] e22
    r3 = e31 [+|...] e32
    assert r1 == e21 || ... || r1 == e32
    assert r2 == e31 || r2 == e32
    return r3
```
anatomy of a metasketch (2)

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

2. **cost function** \((\kappa)\)
   - \(\kappa : \{ P \mid P \in S_i \land S_i \in \mathcal{S} \} \to \mathbb{R}\)
   - assigns a numeric cost to each candidate

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

\(\mathcal{S}, \preceq\) (SSA with iterative deepening)
anatomy of a metasketch (2)

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

2. cost function ($\kappa$)
   - $\kappa : \{ P \mid P \in S_i \land S_i \in \mathcal{S} \} \rightarrow \mathbb{R}$
   - assigns a numeric cost to each candidate

3. gradient function ($g$)

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

Counts the number of defined variables in the candidate program $P$. 

$\mathcal{S}$, $\preceq$ (SSA with iterative deepening)
anatomy of a metasketch (2)

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

2. **cost function** ($\kappa$)
   - $\kappa : \{ P \mid P \in S_i \land S_i \in \mathcal{S} \} \rightarrow \mathbb{R}$
   - assigns a numeric cost to each candidate

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

The cost function $\kappa$ can be *static* (based on program syntax) or *dynamic* (based on runtime behavior).

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

Counts the number of defined variables in the candidate program $P$. 

$\mathcal{S}, \preceq$ (SSA with iterative deepening)
anatomy of a metasketch (2)

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

2. **cost function** ($\kappa$)
   - $\kappa : \{ P \mid P \in S_i \land S_i \in \mathcal{S} \} \to \mathbb{R}$
   - assigns a numeric cost to each candidate

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

The cost function $\kappa$ can be static (based on program syntax) or dynamic (based on runtime behavior).

Any cost function $\kappa$ can be used as long as the result of evaluating $\kappa$ on a program $P$ (and possibly its inputs) can be expressed as a term in a decidable theory.

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

Counts the number of defined variables in the candidate program $P$. 

$\mathcal{S}$, $\preceq$ (SSA with iterative deepening)
anatomy of a metasketch (3)

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 all sketches in \(\mathcal{S}\) that may contain a program \(P\) with \(\kappa(P) < c\)

\(\mathcal{S}, \preceq\) (SSA with iterative deepening)

\(\kappa(P) = i\) for \(P \in S_i \in \mathcal{S}\)
anatomy of a metasketch (3)

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 all sketches in \(\mathcal{S}\) that 
     may contain a program \(P\) with \(\kappa(P) < c\)

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

\[\mathcal{S}, \preceq \text{ (SSA with iterative deepening)}\]

\[\kappa(P) = i \text{ for } P \in S_i \in \mathcal{S}\]
anatomy of a metasketch (3)

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 all sketches in $\mathcal{S}$ that may contain a program $P$ with $\kappa(P) < c$

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

$$\mathcal{S}, \preceq \text{ (SSA with iterative deepening)}$$

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

$$\kappa(P) = i \text{ for } P \in S_i \in \mathcal{S}$$
anatomy of a metasketch (3)

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 all sketches in \(\mathcal{S}\) that *may* contain a program \(P\) with \(\kappa(P) < c\)

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

It is always sound to return the trivial gradient \(g(c) = \mathcal{S}\).

\(\mathcal{S}, \preceq\) (SSA with iterative deepening)

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

\[g(c) = \{ S_i \in \mathcal{S} \mid i < c \}\]
basic idea: two cooperating search algorithms

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

- global search
- local search
- local search
- local search
basic idea: two cooperating search algorithms

Global optimizing search coordinates the activities of local searches running in parallel on individual sketches in $\mathcal{S}$. 
basic idea: two cooperating search algorithms

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

global search

Local combinatorial search employs an incremental form of CEGIS to incorporate the information sent by the global search.

Global optimizing search coordinates the activities of local searches running in parallel on individual sketches in \( \mathcal{S} \).
basic idea: two cooperating search algorithms

Global optimizing search coordinates the activities of local searches running in parallel on individual sketches in $\mathcal{S}$.

Local combinatorial search employs an incremental form of CEGIS to incorporate the information sent by the global search.
basic idea: two cooperating search algorithms

Global optimizing search coordinates the activities of local searches running in parallel on individual sketches in $\mathcal{S}$.

Local combinatorial search employs an incremental form of CEGIS to incorporate the information sent by the global search.
synapse by example

\[ S, \leq, K, g \]

global search

local search
local search
local search

\[ S_1, S_2, S_3, S_4, S_5, S_6 \]
synapse by example
synapse by example

$\mathcal{S}, \preceq, \kappa, g$

local search

global search

local search

local search

UNSAT

$S_1$

$S_3$

$S_4$

$S_5$

$S_6$
synapse by example

$\mathcal{S}, \leq, \kappa, g$

global search

local search

local search

local search

$S_1$

$S_4$

$S_3$

$S_5$

$S_6$
synapse by example
synapse by example
synapse by example

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

global search

\[ \kappa(P) \]

local search

Prune local search spaces using \( \kappa(P) \).
synapse by example

Prune the global search space to $g(\kappa(P))$.

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

$S, \preceq, \kappa, g$
synapse by example

Continues until all remaining sketches return UNSAT and a globally optimal solution is found.
Continues until all remaining sketches return UNSAT and a globally optimal solution is found. Terminates if $g(c)$ is a finite set of sketches for all $c$. 
synapse can solve new classes of problems
synapse can solve new classes of problems

Finds the cheapest program that approximates a given reference program with respect to an application-specific error bound.
**synapse can solve new classes of problems**

- **Parrot**

  *Finds the cheapest program that approximates a given reference program with respect to an application-specific error bound.*

  ```python
  def inversek2j(float x, float y):
    th2 = acos(((x*x) + (y*y) - 0.5) / 0.5)
    th1 = asin((y * (0.5 + 0.5*cos(*th2)) - 0.5*x*sin(*th2)) / (x*x + y*y))
    return th1
  ```

**Solving time (secs)**
synapse solves standard benchmarks optimally

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Solving time (secs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>arraysearch-2</td>
<td></td>
</tr>
<tr>
<td>arraysearch-3</td>
<td></td>
</tr>
<tr>
<td>arraysearch-4</td>
<td></td>
</tr>
<tr>
<td>arraysearch-5</td>
<td></td>
</tr>
<tr>
<td>arraysearch-6</td>
<td></td>
</tr>
<tr>
<td>arraysearch-7</td>
<td></td>
</tr>
<tr>
<td>arraysearch-8</td>
<td></td>
</tr>
<tr>
<td>arraysearch-9</td>
<td></td>
</tr>
<tr>
<td>arraysearch-10</td>
<td></td>
</tr>
<tr>
<td>arraysearch-11</td>
<td></td>
</tr>
<tr>
<td>arraysearch-12</td>
<td></td>
</tr>
<tr>
<td>arraysearch-13</td>
<td></td>
</tr>
<tr>
<td>arraysearch-14</td>
<td></td>
</tr>
<tr>
<td>arraysearch-15</td>
<td></td>
</tr>
</tbody>
</table>

![Array Search Graph](image)
synapse solves standard benchmarks optimally

Array Search

Solving time (secs)

Synapse: 349 bytes
SyGuS: 7.1 MB
is this a cat?
synapse can reason about complex costs
synapse can reason about complex costs

\[ \mathcal{S}, \preceq : \text{a finite set of network topologies} \]

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

\[ g(c) = \mathcal{S} \]
\(\mathcal{S}, \leq, \kappa, g\)

http://synapse.uwplse.org

is this a cat? yes!