Program Synthesis by Sketching

Rastislav Bodik  UC Berkeley

Gilad Arnold, Bob Brayton, Chris Jones, Alan Mishchenko, Armando Solar-Lezama, Koushik Sen, Sanjit Seshia, Liviu Tancau, UC Berkeley  Rodric Rabbah MIT Kemal Ebcioglu, Vijay Saraswat, Vivek Sarkar, Martin Vechev, Eran Yahav  IBM Mooly Sagiv Tel Aviv University
Synthesis

The promise: Automate program development
Abstraction Vs. Performance tradeoff

Level of abstraction

Performance

The performance gap widens

Parallelism

Programming here gets harder

Machine code

Python

Java

C
Software synthesis could help

This transition requires insight (becomes intractable without it)
Software synthesis could help

DSLs:
- StreamIt
- AutoBayes

Generators
- FFTW

Synthesizers
- PRL
- KIDS

- hard-coded heuristics
- domain specific transformations

- interactive proofs
- domain theories

Axiomatic definitions of Types and functions that may be used in the solution
Ex: merge(c, d, b) \iff (\text{ordered}(c) \land \text{ordered}(d)) \Rightarrow (\text{perm}(c \cdot d, b) \land \text{ordered}(b))
Software synthesis could help

- hard-coded heuristics
- domain specific transformations

It’s hard for users to help
- hacking the synthesizer is very hard
- writing domain theories is very hard
The Challenge
Challenge

Establish a synergy between synthesizer and user

Insight
Big picture
Strategy

Exhaustive exploration
Details
Tactics
Research problems

Programmers must be able to contribute expertise
  - How can the programmer contribute insights?
  - How can we use these insights to make synthesis scalable?

Programmers want control over the implementation
  - Can we avoid exposing programmers to synthesizer’s internals?

Programmers insight will sometimes be wrong
  - How can we support debugging
Sketching

Our answer to the challenges of practical synthesis

Key Contributions:
- Synthesis from partial programs
  - PLDI 05
- Design of the SKETCH language
  - ASPLOS 06
- SKETCH Synthesis algorithm
  - ASPLOS 06
- Generalization to Concurrent Programs
  - PLDI 08
- Domain Specific Sketching through Program Reduction
  - PLDI 07
The Sketching Approach

What is sketching

SKETCH Language

SKETCH Synthesis Algorithm

Concurrency

Future Work
Key Observation

Insight and Mechanics are both reflected in the source code

The Sketch solution:

- Write only the code corresponding to insight
- Let the synthesizer derive the mechanics
Merge sort: first, by hand

```c
int[] mergeSort (int[] input, int n) {
    if ( n == 1 ) return input;
    return merge( mergeSort (input[0:n/2-1], n/2),
                   mergeSort (input[n/2:n-1], n-n/2), n);
}

int[] merge (int[] a, int b[], int n) {
    int j=0; int k=0;
    for (int i = 0; i < n; i++)
        if ( a[j] < b[k] )
            result[i] = a[j++];
        else
            result[i] = b[k++];
    return result;
}
```

looks simple to code, but there is a bug
**Merge sort: corrected, by hand**

```c
int[] mergeSort (int[] input, int n) {
    if ( n == 1 ) return input;
    return merge( mergeSort (input[0:n/2-1], n/2),
                   mergeSort (input[n/2:n-1], n-n/2), n);
}

int[] merge (int[] a, int b[], int n) {
    int j=0; int k=0;
    for (int i = 0; i < n; i++)
        if ( j<n/2 && ( !(k<n-n/2) || a[j]<b[k]) )
            result[i] = a[j++];
        else
            result[i] = b[k++];
    return result;
}
```
Merge sort: sketched

```java
int[] mergeSort (int[] input, int n) {
    if (n == 1) return input;
    return merge(mergeSort(input[0:n/2-1], n/2), mergeSort(input[n/2:n-1], n-n/2), n);
}

int[] merge (int[] a, int b[], int n) {
    int j=0; int k=0;
    for (int i = 0; i < n; i++)
        if (hole)
            result[i] = a[j++];
        else
            result[i] = b[k++];
    return result;
}
```
int[] mergeSort (int[] input, int n) {
    if (n == 1)
        return input;
    return merge(
        mergeSort (input[0:n/2-1], n),
        mergeSort (input[n/2:n-1], n-n/2), n);
}

int[] merge (int[] a, int b[], int n) {
    int j=0; int k=0;
    for (int i = 0; i < n; i++)
        if (hole)
            result[i] = a[j++];
        else
            result[i] = b[k++];
    return result;
}

int[] sort (int[] input, int n) {
    for (int i=0; i<n; ++i)
        for (int j=i+1; j<n; ++j)
            if (input[j] < input[i])
                swap(input, j, i);
    return input;
}
The sketching experience

spec + sketch → implementation (completed sketch)
int[] sort (int[] input, int n) {
    for (int i=0; i<n; ++i)
        for (int j=i+1; j<n; ++j)
            if (input[j] < input[i])
                swap(input, j, i);
    return input;
}
Merge sort: sketched

```c
int[] mergeSort (int[] input, int n) {
    if ( n == 1 ) return input;
    return merge(   mergeSort (input[0:n/2-1], n/2),
                    mergeSort (input[n/2:n-1], n-n/2), n);
}

int[] merge (int[] a, int b[], int n) {
    int j=0; int k=0;
    for (int i = 0; i < n; i++)
        if ( hole )
            result[i] = a[j++];
        else
            result[i] = b[k++];
    return result;
}
```
int[] mergeSort (int[] input, int n) {
    if ( n == 1 ) return input;
    return merge(mergeSort(input[0:n/2-1], n/2), mergeSort(input[n/2:n-1], n-n/2), n);
}

int[] merge (int[] a, int[] b[], int n) {
    int j=0; int k=0;
    for (int i = 0; i < n; i++)
        if ( expr(<,&&,||,!,-,[])(a, b, j, k, n, n/2 ) )
            result[i] = a[j++];
        else
            result[i] = b[k++];
    return result;
}
Merge sort: synthesized

```java
int[] mergeSort (int[] input, int n) {
    if (n == 1) return input;
    return merge(mergeSort(input[0:n/2-1], n),
                  mergeSort(input[n/2:n-1], n-n/2), n);
}

int[] merge (int[] a, int b[], int n) {
    int j=0; int k=0;
    for (int i = 0; i < n; i++)
        if (j<n/2 && ( !(k<n-n/2) || a[j]<b[k]) )
            result[i] = a[j++];
        else
            result[i] = b[k++];
    return result;
}
```
The SKETCH Language
PLDI 05, ASPLOS 06

What is sketching

SKETCH Language

SKETCH Synthesis Algorithm

Concurrency

Future Work
Language design goals

Learnable
- Embed easily into a known language

Expressive
- allow programmer to express different insights about holes

Induces a simple synthesis problem
- ideally, domain independent
**SKETCH: two simple constructs**

**spec:**
```c
int foo (int x) {
    return x + x;
}
```

**sketch:**
```c
int bar (int x) implements foo {
    return x << ??;
}
```

**result:**
```c
int bar (int x) implements foo {
    return x << 1;
}
```

**Assertions can also be used to define behavior**
Case study 1: Silver Medal in a SKETCH contest

The 4x4-matrix transpose, the specification:

```c
int[16] trans(int[16] M) {
    int[16] T = 0;
    for (int i = 0; i < 4; i++)
        for (int j = 0; j < 4; j++)
            T[4 * i + j] = M[4 * j + i];
    return T;
}
```

Implementation idea: parallelize with SIMD
Intel shufps SIMD instruction

SHUFP (shuffle parallel scalars) expressed in SKETCH:
The SIMD matrix transpose, sketched

```
int[16] trans_sse(int[16] M) implements trans {
  int[16] S = 0, T = 0;
  repeat (??) S[??::4] = shufps(M[??::4], M[??::4], ??);
  repeat (??) T[??::4] = shufps(S[??::4], S[??::4], ??);
  return T;
}

int[16] trans_sse(int[16] M) implements trans { // synthesized code
  S[4::4] = shufps(M[6::4], M[2::4], 11001000b);
  S[0::4] = shufps(M[11::4], M[6::4], 10010110b);
  S[12::4] = shufps(M[0::4], M[2::4], 10001101b);
  S[8::4] = shufps(M[8::4], M[12::4], 11010111b);
  T[4::4] = shufps(S[11::4], S[1::4], 10111100b);
  T[12::4] = shufps(S[3::4], S[8::4], 11000011b);
  T[8::4] = shufps(S[4::4], S[9::4], 11100010b);
  T[0::4] = shufps(S[12::4], S[0::4], 10110100b);
}
```

From the contestant email: Over the summer, I spent about 1/2 a day manually figuring it out. Synthesis time: 30 minutes.
Beyond synthesis of constants

Sometimes the insight is “I want to complete the hole with an of particular syntactic form.”

- Array index expressions: \[ A[ ???i+??*j+?? ] \]

- Polynomial of degree 2 over x: \[ ???*x*x + ???*x + ?? \]

Primitive holes can be used synthesize arbitrary expressions, statements, …

- we also can make these “generators” reusable
Reusable expression generators

Following function synthesizes to one of $a$, $b$, $a+b$, $a-b$, $a+b+a$, ...

```c
inline int expr(int a, int b){ // generator
    switch(??) {
        case 0: return a;
        case 1: return b;
        case 2: return expr(a,b) + expr(a,b);
        case 3: return expr(a,b) - expr(a,b);
    }
}
```
Synthesizing polynomials

```c
int spec (int x) {
    return 2*x*x*x*x*x + 3*x*x*x*x + 7*x*x + 10;
}

int p (int x) implements spec {
    return (x+1)*(x+2)*poly(3,x);
}

inline int poly(int n, int x) {
    if (n==0) return ??
    else return x * poly(n-1, x) + ??
}
```

Notice the absence of any meta-variables. The generator `poly()` is an ordinary function.

Here, SKETCH performs polynomial division. Result of division is what `poly(3,x)` is synthesized into.
Syntactic Sugar

Easy to add new constructs as syntactic sugar

- `reorder{ s1; s2; … ; sn; }

- `x = { | (a | b | c)(.next)? |}
The SKETCH Synthesis Algorithm

What is sketching

SKETCH Language

SKETCH Synthesis Algorithm

Concurrency

Future Work
Sketch is a set of programs

A sketch syntactically describes a set of candidate programs.
- The ?? operator is modeled as a special input, called control:

  ```c
  bit[W] isolSk(bit[W] x) {
    return ~(x+??) & (x+??);
  }
  bit[W] isolSk(bit[W] x, int c1, c2) {
    return ~(x+c1) & (x+c2);
  }
  ```

The set is defined in terms of the values of the controls

  \[ S = \{ Sk(c) \text{ where } c \text{ is an assignment to the holes} \} \]
Sketch synthesis = Search

Synthesis reduces to a search for the correct candidate
  - Search for control values satisfying the following equation:

  \[ \exists \ c. \ \forall \ x. \ Spec(x) = Sk(x,c) \]

Adding additional insight reduces the search space
  for the user, adding insight reduces to writing more code
  programmers know how to do that
Inductive Synthesis

Synthesize from a set of observations

A little history
- Algorithmic debugging (Shapiro 1982)
- Inductive logic programming (Muggleton 1991)
- Programming by example (e.g. Lau 1999)

Two big issues
- Convergence: How do you know your solution generalizes
- Efficiency: Deriving a candidate from observations is hard
Convergence

Idea: Couple Inductive synthesizer with a verifier
   - Verifier is charged with detecting convergence

Counterexamples make great empirical observations
   - new counterexample ➔ new information
Convergence

Inductive Synthesizer
- Derive candidate implementation from concrete inputs.

Verifier
- Your verifier goes here

candidate implementation

succeed

fail

buggy

add counterexample input

observation set E

ok

fail
Convergence

Example: remove an element from a doubly linked list.

```c
void remove(list l, node n){
    if (cond(l,n)) { assign(l, n); }
    if (cond(l,n)) { assign(l, n); }
    if (cond(l,n)) { assign(l, n); }
    if (cond(l,n)) { assign(l, n); }
    if (cond(l,n)) { assign(l, n); }
}

int N = 6;
void test(int p){
    nodes[N] nodes;
    list l;
    initialize(l, nodes);  //... add N nodes to list
    remove(l, nodes[p]);
    checkList(nodes, l, p);
}
```
**Ex: Doubly Linked List Remove**

```c
void remove(list l, node n) {
    if(n.prev != l.head) {
        n.next.prev = n.prev;
    }
    if(n.prev != n.next) {
        n.prev.next = n.next;
    }
}
```

**Counterexamples**

<table>
<thead>
<tr>
<th>p = 3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

42
Ex: Doubly Linked List Remove

void remove(list l, node n) {
    if(n.prev != null)
        n.next.prev = n.prev;

    if(l.head == n)
        l.head = n.next;

    l.tail = l.tail;

    if(l.head!=n.next)
        n.prev.next = n.next;
}

<table>
<thead>
<tr>
<th>Counterexamples</th>
</tr>
</thead>
<tbody>
<tr>
<td>p = 3</td>
</tr>
<tr>
<td>p = 0</td>
</tr>
</tbody>
</table>

Ex: Doubly Linked List Remove

```java
void remove(list l, node n) {
    if(n.prev == null)
        l.head = n.next;
    if(n.next == null)
        l.tail = n.prev;
    if(n.next != l.head)
        n.prev.next = n.next;
    if(n.next != null)
        n.next.prev = n.prev;
}
```

<table>
<thead>
<tr>
<th>Counterexamples</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>p = 3</td>
<td></td>
</tr>
<tr>
<td>p = 0</td>
<td></td>
</tr>
<tr>
<td>p = 5</td>
<td></td>
</tr>
</tbody>
</table>

Process takes < 1 second
The number of counterexample inputs is small.
The number of counterexample inputs is small

Correlation between number of iterations and number of bits of unknowns

\[ y = 0.0644x + 9.7299 \]

\[ R^2 = 0.9457 \]
Inductive Synthesis

Deriving a candidate from a set of observations

Key:
- Frame as a constraint satisfaction problem
- Avoid enumeration; use algebraic techniques instead

Encode candidate space as a bit-vector
- Natural encoding given the integer holes

Encode synthesis as boolean constraints on bit-vector

\[ \exists \ c. \ \forall \ x \in \mathcal{E}. \ Spec(x) = Sk(x, c) \]
where \( \mathcal{E} = \{x_1, x_2, \ldots, x_k\} \)

Solve constraints using SAT solver
## Interesting Benchmarks

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Unknowns</th>
<th>Solution Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES</td>
<td>32769 bits</td>
<td>78.8 min</td>
</tr>
<tr>
<td>SSE Matrix Transpose</td>
<td>24 integers + 64 bits</td>
<td>30 min</td>
</tr>
<tr>
<td>CRC</td>
<td>8192 bits</td>
<td>13.5 min</td>
</tr>
<tr>
<td>Doubly Linked List Remove</td>
<td>60 bits</td>
<td>&lt; 1 sec</td>
</tr>
<tr>
<td>Enqueue</td>
<td>271 int &amp; bit</td>
<td>1 min</td>
</tr>
<tr>
<td>16 bit Morton Numbers</td>
<td>332 int &amp; bit</td>
<td>20 min</td>
</tr>
<tr>
<td>32 bit fast parity</td>
<td>45 bits</td>
<td>11 sec</td>
</tr>
<tr>
<td>Sort (bounded)</td>
<td>363 int &amp; bit</td>
<td>3.5 min</td>
</tr>
</tbody>
</table>
Concurrency

What is sketching

SKETCH Language

SKETCH Synthesis Algorithm

Concurrency

Future Work
Concurrent programs

Ex: Concurrent Enqueue using AtomicSwap

```java
class Queue {
    QueueEntry prevHead = new QueueEntry(null);
    QueueEntry tail = prevHead;

    void Enqueue(Object newobject) {
        Node tmp = null;
        newEntry = new QueueEntry(newobject);
        tmp = AtomicSwap(tail, newEntry);
        tmp.next = newEntry;
    }
}
```

```java
Object AtomicSwap(ref Object loc, Object entry) atomic {
    Object old = loc;
    loc = entry;
    return old;
}
```
Generalization to Parallelism

Output now dependent on thread interleaving
- Make interleaving schedule part of the observations
- Most verifiers can provide a counterexample trace

Using the observations becomes harder
- Schedule generated as witness for a given candidate
- How do we use it to rule out other incorrect candidates?