CSC410
Program Synthesis

AZADEH FARZAN
FALL 2020
The idea

Produce small code fragments that satisfy the given specification

As a programming aid:
- It helps you write programs.
- The insight is yours, the tedious work is done by the tool.
Let’s start with a concrete example ...
Sketching
Problem Setup

Fundamentals:
- How does the tool define a universe of "things" that can fill up a hole?
- How does the programmer communicate what he wants?

Sketches:
- Programs with holes.
- Write what you know, use the holes for the details you do not want to fully figure out.
Program Synthesis Problem

**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;
}
```
Specification

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

We can use *equivalence to existing code* as specification.

We can also use *logical specification*, exactly as in verification.

```c
spec: assert r == 2 * x
```

We can also use *samples of input/out pairs* as specification.

```c
spec: (r,x): (0,0) (2,1) (6,3) (20, 10)
```
**Sketch**

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

Holes are meant to be placeholder for code fragments.

Fragments range over a user-defined set.

---

Defining a good set of fragments is the key to effective Sketching!
**Sketch**

**Fragments** range over a user-defined set.

**Sets** are defined hierarchically.

```
Sets of Integer Constants
  Sets of Expressions
    Sets of Statements
      Sets of Procedures
        Sketch
```

Sets are defined hierarchically.
Stepping back to a general setting ...
What is program synthesis?

Say we have a property \( \phi \) over input/output pairs for a program.

We can formally frame the problem of finding a function that fits this specification through:

\[
\exists P. \forall x. \phi(x, P(x))
\]

When does automated programming make sense?

- when it is easier to write \( \phi \) than it is to write \( P \).
- when it is easier to be convinced about \( \phi \) than it is to write \( P \).
What is program synthesis?

Say we have a property \( \phi \) over input/output pairs for a program. We can formally frame the problem of finding a function that fits this specification through:

\[
\exists P. \forall x. \phi(x, P(x))
\]

Generally, this is complex constraint solving problem, where generic efficient solutions do not exist.

Algorithmic challenge: solve this quantified satisfiability problem by reducing it to a flat satisfiability problem for which efficient solvers exist.
Extended Example
Example

Least significant zero bit: 0010 0101 \rightarrow 0000 0010

Trivial Code:

```c
int W = 32;
bit[W] isolate0(bit[W] x) { // W: word size
    bit[W] ret = 0;
    for (int i = 0; i < W; i++)
        if (!x[i]) { ret[i] = 1; return ret; }
}
```

Insight: adding 1 to a string of 1's turns the next 0 to 1.
Trivial code:

```c
int W = 32;
bit[W] isolate0(bit[W] x) { // W: word size
    bit[W] ret = 0;
    for (int i = 0; i < W; i++)
        if (!x[i]) { ret[i] = 1; return ret; }
}
```

Sketch of better code:

```c
bit[W] isolateSk (bit[W] x) implements isolate0 {
    return !(x + ??) & (x + ??);
}
```

least significant zero bit: 0010 0101 → 0000 0010

!(x + 0) & (x + 1)
Example

Assume all we know is that the solution uses $x$, $+$, $\&$, and $!$.

Looser Sketch:

```plaintext
bit[W] tmp=0;
{x | tmp |} = {(!)?((x | tmp) (& | +) (x | tmp | ??)) |};
{x | tmp |} = {(!)?((x | tmp) (& | +) (x | tmp | ??)) |};
return tmp;
```

Regular expressions are used to define expressions and choice in code fragments.

Even more expressive:

```plaintext
bit[W] tmp=0;
repeat(3){
   {x | tmp |} = {(!)?((x | tmp) (& | +) (x | tmp | ??)) |};
}
return tmp;
```