Lecture 10
Unbounded Verification

Nadia Polikarpova
(some material from Peter Müller, ETH Zurich)

Logistics

Projects

- I expect you to meet with me at least once before the presentation to discuss the progress
- Book a date through the spreadsheet
  - During Office Hours
  - Two teams per week

Map of the unit

Constraint-based synthesis

- How to solve constraints about infinitely many inputs? CEGIS
- How to encode semantics of looping / recursive programs?
  - Bounded reasoning
  - Unbounded / deductive reasoning

Enumerative (and deductive) synthesis

- How to use deductive reasoning to guide the search?

Constraint-based synthesis from specifications

Behavioral constraints
= assertions, reference implementation, pre/post

Structural constraints

$\xrightarrow{\text{encoding}}$

$$\exists c\,.\,\forall x\,.\,Q(c, x)$$

Why is this hard?

```
gcd (int a, int b) returns (int x)
  requires a > 0 ∧ b > 0
  ensures  a % x = 0 ∧ b % x = 0
           ∀c . 0 < c < x ⇒ a % c ≠ 0 ∨ b % c ≠ 0
{
  int x , y := a, b;
  while (x != y) {
    if (x > y) x := ??*x + ??*y + ??;
    else y := ??*x + ??*y + ??;
}}
```

infinitely many inputs

infinitely many paths!

Until now

To encode a program as constraints, need to understand what it does

- $\mathcal{A}[\![\cdot]\!] : e \to \Sigma \to \mathbb{Z}$
- $\mathcal{C}[\![\cdot]\!] : c \to \Sigma \to \Sigma$

This is called *denotational semantic*: map a program to a function

- Hard for loops and recursion: need to find a function that satisfies a fixpoint equation
- We had to resort to bounded unrolling ☹

What's wrong with unrolling?

```
gcd (int a, int b) returns (int x)
  requires a > 0 ∧ b > 0
  ensures  a % x = 0 ∧ b % x = 0
           ∀c . 0 < c < x ⇒ a % c ≠ 0 ∨ b % c ≠ 0
{
  int x , y := a, b;
  while (x != y) {
    if (x > y) x := ??*x + ??*y + ??;
    else y := ??*x + ??*y + ??;
}}
```

Unroll with depth = 1

```
if (x != y) {
  if (x > y)
    x := ??*x + ??*y + ??;
  else
    y := ??*x + ??*y + ??;
  assert !(x != y);
}
```

Unsatisfiable sketch

What's wrong with unrolling?

What if inputs are 2-bit words?

```
gcd (int a, int b) returns (int x)
  requires a > 0 ∧ b > 0
  ensures  a % x = 0 ∧ b % x = 0
           ∀c . 0 < c < x  ⇒  a % c ≠ 0 ∨ b % c ≠ 0
{
  int x , y := a, b;
  while (x != y) {
    if (x > y) x := ??*x + ??*y + ??;
    else y := ??*x + ??*y + ??;
}}
```

Unroll with depth = 1

Unsound solution!

```
if (x != y) {
  if (x > y)
    x := 0 *x + 0 *y + 1 ;
  else
    y := 0 *x + 0 *y + 1 ;
  assert !(x != y);
}
```

Constraint-based synthesis from specifications

Behavioral constraints
= assertions, reference implementation, pre/post

Structural constraints

$$\xrightarrow{\text{encoding}} \quad \exists c\,.\,\forall x\,.\,Q(c,x)$$

If we want to synthesize programs that are correct on all inputs, we need a better way to deal with loops!

Axiomatic semantics

[Hoare '69]

Instead of asking "What does this program do?", ask "Does this property hold?"

- $\mathcal{A}[\![\cdot]\!] : e \to \Sigma \to \mathbb{Z}$
- $\cancel{\mathcal{C}[\![\cdot]\!] : c \to \Sigma \to \Sigma}$
- $\{P\}\ c\ \{Q\}$ ("if P holds before executing c, then Q holds after")

Axiomatic semantics = program logic

- A set of rules for proving *judgments* about programs

Hoare logic = a program logic for simple imperative programs

The Imp language

```
e ::= n | x |
      e + e | e - e | e * e |
      e = e | e < e | !e | e && e
c ::= skip                  |
      x := e                |
      c ; c                 |
      if e then c else c    |
      while e do c
```

Hoare triples

Properties of programs are specified as judgments

$$\{P\}\ c\ \{Q\}$$

where c is a command and $P, Q: \sigma \to Bool$ are predicates

- e.g. if $\sigma = [x \mapsto 2]$ and $P \equiv x > 0$ then $P\ \sigma = \mathrm{T}$

Terminology

- Judgments of this kind are called *(Hoare) triples*
- P is called precondition
- Q is called postcondition

Meaning of triples

The meaning of $\{P\}\ c\ \{Q\}$ is:

- **if** P holds in the initial state σ, and
- **if** the execution of c from σ terminates in a state σ'
- **then** Q holds in σ'

This interpretation is called *partial correctness*

- termination is not essential

Another possible interpretation: *total correctness*

- **if** P holds in the initial state σ
- **then** the execution of c from σ terminates in a state (call it σ')
- **and** Q holds in σ'

Example: swap

$\{\top\}$

```
x := x + y; y := x - y; x := x - y
```

$\cancel{\{x = y \wedge y = x\}}$

We have to express that y in the final state is equal to x in the initial state!

Logical variables

$\{\mathrm{x} = N \wedge \mathrm{y} = M\}$

```
x := x + y; y := x - y; x := x - y
```

$\{\mathrm{x} = M \wedge \mathrm{y} = N\}$

Assertions can contain *logical variables*

- may occur only in pre- and postconditions, not in programs
- the state maps logical variables to their values, just like normal variables

Inference system

We formalize the semantics of a language by describing which judgments are valid about a program

An *inference system*

- a set of *axioms* and *inference rules* that describe how to derive a valid judgment

We combine axioms and inference rules to build *inference trees* (derivations)

Semantics of skip

`skip` does not modify the state

$$\{\, P \,\}\ \texttt{skip}\ \{\, P \,\}$$

Semantics of assignment

x := e assigns the value of e to variable x

$$\{\, P[x \mapsto e] \,\}\ \ \texttt{x} \coloneqq \texttt{e}\ \ \{\, P \,\}$$

- Let σ be the initial state
- Precondition: $(P[x \mapsto e])\sigma$, i.e., $P(\sigma[x \mapsto \mathcal{A}[\![e]\!]\sigma])$
- Final state: $\sigma' = \sigma[x \mapsto \mathcal{A}[\![e]\!]\sigma]$
- Consequently, P holds in the final state

Semantics of composition

Sequential composition `c1 ; c2` executes **c1** to produce an intermediate state and from there executes **c2**

$$\frac{\{P\}\ c_1\ \{R\} \qquad \{R\}\ c_2\ \{Q\}}{\{P\}\ c_1; c_2\ \{Q\}}$$

Example: swap

inference tree

leaves = axioms

$$\text{assign}\ \dfrac{}{\{x = N + M \land y = N\}\ \texttt{x := x - y}\ \{x = M \land y = N\}}$$

$$\text{assign}\ \dfrac{}{\{x = N + M \land y = M\}\ \texttt{y := x - y}\ \{x = N + M \land y = N\}}$$

edges = rules

$$\text{comp}\ \dfrac{}{\{x = N + M \land y = M\}\ \texttt{y := x - y; x := x - y}\ \{x = M \land y = N\}}$$

$$\dfrac{}{\{x = N \land y = M\}\ \texttt{x := x + y}\ \{x = N + M \land y = M\}}\ \text{assign}$$

$$\dfrac{}{\{x = N \land y = M\}\ \texttt{x := x + y; y := x - y; x := x - y}\ \{x = M \land y = N\}}\ \text{comp}$$

root = triple to prove

Proof outline

An alternative (more compact) representation of inference trees

$\{x = N \land y = M\}$

$\Rightarrow$

$\{(x + y) - ((x + y) - y) = M \land (x + y) - y = N\}$

```
x = x + y;
```

$\{x - (x - y) = M \land x - y = N\}$

```
y = x - y;
```

$\{x - y = M \land y = N\}$

```
x = x - y
```

$\{x = M \land y = N\}$

Rule of consequence

$$\frac{\{P'\}\ c\ \{Q'\}}{\{P\}\ c\ \{Q\}}\quad \text{if}\quad P \Rightarrow P' \land Q' \Rightarrow Q$$

Corresponds to adding $\Rightarrow$ steps in a proof outline

Here $R \Rightarrow S$ should be read as

- "We can prove for all states σ, that $R\ \sigma$ implies $S\ \sigma$"

Semantics of conditionals

$$\frac{\{P \land e\}\ c_1\ \{Q\} \qquad \{P \land \neg e\}\ c_2\ \{Q\}}{\{P\}\ \texttt{if}\ e\ \texttt{then}\ c_1\ \texttt{else}\ c_2\ \{Q\}}$$

Example: absolute value

$\{\top\}$

```
if x < 0 then
```

$\{x < 0\}$

$\Rightarrow$

$\{-x \geq 0\}$

```
    x := -x
```

$\{x \geq 0\}$

```
else
```

$\{\neg(x < 0)\}$

$\Rightarrow$

$\{x \geq 0\}$

```
    skip
```

$\{x \geq 0\}$

$\{x \geq 0\}$

$$\frac{\{P \wedge e\}\ c_1\ \{Q\} \qquad \{P \wedge \neg e\}\ c_2\ \{Q\}}{\{P\}\ \texttt{if}\ e\ \texttt{then}\ c_1\ \texttt{else}\ c_2\ \{Q\}}$$

Semantics of loops

We want to say:

- P holds initially
- after executing c
  - if e still holds, we execute it c again
  - otherwise, Q holds

$$\frac{\{?\}\ c\ \{?\}}{\{P\}\ \texttt{while}\ e\ \texttt{do}\ c\ \{Q\}}$$

Semantics of loops

loop invariant

$$\frac{\{I \land e\}\ c\ \{I\}}{\{I\}\ \texttt{while}\ e\ \texttt{do}\ c\ \{\neg e \land I\}}$$

Example: GCD

$\{x = N \wedge y = M \wedge N > 0 \wedge M > 0\}$

$\Rightarrow$

$\{I\}$

```
while x != y do
```

$\{I \wedge x \neq y\}$

```
    if x > y then
      x := x - y
    else
      y := y - x
```

$\{I\}$

$\{I \wedge x = y\}$

$\Rightarrow$

$\{x = \text{gcd}(N, M)\}$

Guessing the loop invariant:

x	y
10	4
6	4
2	4
2	2

$$I \equiv \text{gcd}(x, y) = \text{gcd}(N, M)$$

Example: GCD

$\{x = N \land y = M \land N > 0 \ \land M > 0\}$

$\Rightarrow$

$\{\gcd(x, y) = \gcd(N, M) \land x, y > 0\}$

```
while x != y do
```

$\{\gcd(x, y) = \gcd(N, M) \land x, y > 0 \land \ x \neq y\}$

```
if x > y then
```

$\{\gcd(x, y) = \gcd(N, M) \land \ x \neq y \land x > y\}$

$\Rightarrow$

$\{\gcd(x - y, y) = \gcd(N, M) \land x - y, y > 0\}$

```
x := x - y
```

$\{\gcd(x, y) = \gcd(N, M) \land x, y > 0\}$

```
else
    y := y - x
```

$\{\gcd(x, y) = \gcd(N, M) \land x, y > 0\}$

$\{\gcd(x, y) = \gcd(N, M) \land x, y > 0 \land \ x = y\}$

$\Rightarrow$

$\{x = \gcd(N, M)\}$

Termination

loop variant / ranking function /
termination metric

$$\frac{\{I \wedge e \wedge r = R\}\ c\ \{I \wedge r < R \wedge r \geq 0\}}{\{I\}\ \texttt{while}\ e\ \texttt{do}\ c\ \{\neg e \wedge I\}}$$

Example: GCD

```
while x != y do

    if x > y then
        x := x - y
    else
        y := y - x
```

Example: GCD

$\{x = N \wedge y = M \wedge N > 0 \;\wedge M > 0\}$

$\Rightarrow$

$\{\gcd(x, y) = \gcd(N, M) \wedge x, y > 0\}$

```
while x != y do
```

$\{\gcd(x, y) = \gcd(N, M) \wedge x, y > 0 \wedge x + y = R \wedge \; x \neq y\}$

```
if x > y then
    x := x - y
else
    y := y - x
```

$\{\gcd(x, y) = \gcd(N, M) \wedge x, y > 0 \wedge x + y < R \wedge x + y \geq 0\}$

$\{\gcd(x, y) = \gcd(N, M) \wedge x, y > 0 \wedge \; x = y\}$

$\Rightarrow$

$\{x = \gcd(N, M)\}$

Program verifiers

Dafny demo: SumMax

https://rise4fun.com/Dafny/YyGpL