Automated Program Verification and Testing
15414/15614 Fall 2016
Lecture 14:
Deductive Verification, Part 2

Matt Fredrikson
mfredrik@cs.cmu.edu

October 18, 2016

VC Generation (Review)

Given an assertion Q and program c, we'll describe a function:

- That is a **predicate transformer**: produces another assertion
- Assertion for the corresponding precondition P for c
- P guaranteed to be the **weakest** such assertion

This is the **weakest precondition** predicate transformer $\mathsf{wp}(c, Q)$

The weakest precondition satisfies the following conditions:

1. The triple $[\mathsf{wp}(c, Q)]\ c\ [Q]$ is valid
2. For any P where $[P]\ c\ [Q]$ is valid, $P \Rightarrow \mathsf{wp}(c, Q)$

For partial correctness, use **weakest liberal precondition** $\mathsf{wlp}(c, Q)$

Weakest Liberal Precondition (Review)

$$
\begin{aligned}
\mathsf{wlp}(x := a, Q) &= Q[a/x] \\
\mathsf{wlp}(x[a_1] := a_2, Q) &= Q[a\langle a_1 \triangleleft a_2\rangle/x] \\
\mathsf{wlp}(c_1; c_2, Q) &= \mathsf{wlp}(c_1, \mathsf{wlp}(c_2, Q)) \\
\mathsf{wlp}(\textbf{if } b \textbf{ then } c_1 \textbf{ else } c_2, Q) &= (b \to \mathsf{wlp}(c_1, Q)) \land (\neg b \to \mathsf{wlp}(c_2, Q))
\end{aligned}
$$

Approximate Weakest Precondition (Review)

In general, we can't always compute wlp for loops

Instead, we'll **approximate** it with help from annotations

Now we'll assume loops have the syntax:

$$\textbf{while } b \textbf{ do } \{I\}\ c$$

I is a loop invariant provided by the programmer

The approximate wlp for **while** will still be a valid precondition

But it may not be the weakest precondition: even if

$$\{P\}\ \textbf{while } b \textbf{ do } c\ \{Q\}$$

is valid, it might not be that:

$$P \Rightarrow \mathsf{wlp}(\textbf{while } \{I\}\ b \textbf{ do } c, Q)$$

Verification Conditions: While Loop (Review)

If we define

$$\text{wlp}(\textbf{while}\ \{I\}\ b\ \textbf{do}\ c, Q) = I$$

Then we still need to show that

- $I \wedge \neg b$ establishes Q
- I is a loop invariant

Defining the set of verification conditions,

$$\text{vc}(\textbf{while}\ \{I\}\ b\ \textbf{do}\ c, Q) = \left\{ \begin{array}{l} I \wedge \neg b \Rightarrow Q \\ I \wedge b \Rightarrow \text{wlp}(c, Q) \end{array} \right\}$$

To summarize, for Q to hold after executing a loop:

1. Each formula in $\text{vc}(\textbf{while}\ \{I\}\ b\ \textbf{do}\ c, Q)$ must be valid
2. $\text{wlp}(\textbf{while}\ \{I\}\ b\ \textbf{do}\ c, Q) = I$ must be valid

Propagating Verification Conditions (Review)

while is the only command that introduces new conditions

But other statements might contain loops

Need to define vc for them as well:

- $\text{vc}(x := a, Q) = \varnothing$
- $\text{vc}(c_1; c_2, Q) = \text{vc}(c_1, \text{wlp}(c_2, Q)) \cup \text{vc}(c_2, Q)$
- $\text{vc}(\textbf{if } b \textbf{ then } c_1 \textbf{ else } c_2, Q) = \text{vc}(c_1, Q) \cup \text{vc}(c_2, Q)$

In short, compound statements collect conditions from constituents

Verification Using wlp

Bringing all of this together, we can verify

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

for an annotated program c

1. Compute P' = wlp(c, Q)
2. Compute vc(c, Q)
3. Check validity of $P \to P'$
4. Check validity of each $F \in$ vc(c, Q)

If (3) and (4) pass, then $\{P\}\ c\ \{Q\}$ is valid

If $\{P\}\ c\ \{Q\}$ is valid, then will (3) and (4) pass?
No. Loop invariants might be too weak!

Example

Let's verify the example from last lecture:

$$\begin{array}{l}
\{\textit{true}\} \\
r := x; q := 0; \\
\textbf{while } y \leq r \textbf{ do} \\
\quad r := r - y; q := q + 1 \\
\{r < y \wedge x = r + (q \times y)\}
\end{array}$$

Recall our loop invariant:

$$\begin{array}{l}
\{\textit{true}\} \\
r := x; q := 0; \\
\textbf{while } y \leq r \textbf{ do} \\
\{x = r + (q \times y)\} \\
\quad r := r - y; q := q + 1 \\
\{r < y \wedge x = r + (q \times y)\}
\end{array}$$

Example

Define the following shorthand:

- $c_1 : r := x$
- $c_2 : q := 0$
- $c_3 : r := r - y$
- $c_4 : q := q + 1$
- c_5 : **while** $y \leq r$ **do** $c_3; c_4$

$\{$*true*$\}$
$c_1; c_2;$
while $y \leq r$ **do**
$\{x = r + (q \times y)\}$
$\quad c_3; c_4$
$\{r < y \wedge x = r + (q \times y)\}$

We need to show these are valid:

$$\textit{true} \Rightarrow \text{wlp}(c_1; c_2; c_5, r < y \wedge x = r + (q \times y))$$
$$\text{vc}(c_1; c_2; c_5, r < y \wedge x = r + (q \times y))$$

We'll start with *true* $\Rightarrow$ wlp$(c_1; c_2; c_5, r < y \wedge x = r + (q \times y))$

Example

$$\textit{true} \Rightarrow \mathsf{wlp}(c_1; c_2; c_5, r < y \land x = r + (q \times y))$$

Let's use $Q : r < y \land x = r + (q \times y)$, $I : x = r + (q \times y)$

We begin by applying the rule for composition twice:

$$\mathsf{wlp}(c_1; c_2; c_5, Q) = \mathsf{wlp}(c_1, \mathsf{wlp}(c_2, \mathsf{wlp}(c_5, Q)))$$

This brings us to $\mathsf{wlp}(c_5, Q)$:

$$\mathsf{wlp}(\textbf{while } y \leq r \textbf{ do } \{I\}\ c_3; c_4, Q) = I$$

We also have verification conditions:

$$\mathsf{vc}(c_5, Q) = \{I \land \neg b \Rightarrow Q, I \land b \Rightarrow \mathsf{wlp}(c_3; c_4, Q)\}$$

Example

Let's work out the VC $I \wedge b \Rightarrow \mathsf{wlp}(c_3; c_4, Q)$

We have that:

$$\begin{aligned}\mathsf{wlp}(r := r - y; q := q + 1, Q) &= \mathsf{wlp}(r := r - y, \mathsf{wlp}(q := q + 1, Q)) \\ &= \mathsf{wlp}(r := r - y, Q[q/q + 1]) \\ &= \mathsf{wlp}(r := r - y, r < y \wedge x = r + ((q + 1) \times y)) \\ &= (x = (r - y) + ((q + 1) \times y))\end{aligned}$$

So, we have:

$$\mathsf{vc}(c_5, Q) = \{I \wedge \neg b \Rightarrow Q, I \wedge b \Rightarrow (x = (r - y) + ((q + 1) \times y))\}$$

Example

Recalling that $\mathsf{wlp}(c_5, Q) = I$, we now need $\mathsf{wlp}(c_2, I)$:

$$\begin{aligned} \mathsf{wlp}(q := 0, x = r + (q \times y)) &= (x = r + (0 \times y)) \\ &= x = r \end{aligned}$$

Moving on, our final step is $\mathsf{wlp}(c_1, x = r)$:

$$\mathsf{wlp}(r := x, x = r) = (x = x)$$

Popping back to our top-level procedure:

1. Compute $P' = \mathsf{wlp}(c, Q)$
   $P' = (x = x)$
2. Compute $\mathsf{vc}(c, Q)$
   $\mathsf{vc}(c, Q) = \{I \wedge \neg b \Rightarrow Q, I \wedge b \Rightarrow (x = (r - y) + ((q + 1) \times y))\}$
3. Check validity of $P \rightarrow P'$
   Clearly, *true* $\Rightarrow (x = x)$
4. Check validity of each $F \in \mathsf{vc}(c, Q)$

Example

Check validity of each $F \in \mathsf{vc}(c, Q)$:

$$\mathsf{vc}(c, Q) = \left\{ \begin{array}{l} x = r + (q \times y) \land \neg(y \leq r) \Rightarrow r < y \land x = r + (q \times y) \\ x = r + (q \times y) \land y \leq r \Rightarrow (x = (r - y) + ((q + 1) \times y)) \end{array} \right\}$$

The first is true because $\neg(y \leq r) \Leftrightarrow r < y$

The second we get by algebraic calculation

Therefore, the triple is valid

$$\begin{array}{l} \{\textit{true}\} \\ r := x; q := 0; \\ \textbf{while } y \leq r \textbf{ do} \\ \quad r := r - y; q := q + 1 \\ \{r < y \land x = r + (q \times y)\} \end{array}$$

Imp: New Features

Now we'll add two new featues to our language:

- Assertion annotations
- Procedure

Assertion annotations take the form:

$$\{P\}$$

Semantically, treat them like runtime assertions

Execution halts if the expression isn't true in current environment

Think of assertions as **formal comments**

Imp: Procedures

$$\begin{array}{l}\textbf{proc}\ \text{LinearSearch}(a : \textit{array}, l : \textit{int}, u : \textit{int}, e : \textit{int}) \\ \quad \textbf{requires}\ 0 \leq l \wedge u < |a| \\ \quad \textbf{ensures}\ (\textit{rv} = 1) \leftrightarrow (\exists i. l \leq i \leq u \wedge a[i] = e)\end{array}$$

We'll consider programs with annotated procedures

- **Precondition** annotated with **requires**
- **Postcondition** annotated with **ensures**
- Free variables in the pre- and postconditions can be formal parameters
- Postcondition can also include special variable *rv*
- *rv* stands for the return value

Proving Partial Correctness

Now, a program is partially correct if for each procedure P:

1. Whenever P's preconditions are satisfied on entry
2. P's postconditions are satisfied on exit

We'll extend the approach we've talked about so far

1. Reduce the annotated program to a set of verification conditions
2. If all VCs are valid, then the program is correct

Our approach will be different:

- Use annotations to decompose the program into simpler parts
- Generate VC for each part in isolation, assuming each annotation holds
- Make sure that correctness of the whole follows from correctness of each part

Basic Paths

A **basic path** is a sequence of instructions that:

- Begins at procedure precondition or loop invariant
- Ends at a loop invariant, assertion, or procedure postcondition
- Doesn't cross loops: invariants only at beginning or end of path

Basic paths correspond to straight-line segments of code

Think of a Hoare triple over a sequence command:

$$\{P\}\ c_1; c_2; \ldots; c_n\ \{Q\}$$

P, Q are pre-/postconditions, loop invariants, or assertion guards

Basic Paths: Example

$$
\begin{array}{l}
\textbf{proc } \text{LinearSearch}(a : \textit{array}, l, u, e) \\
\quad \textbf{pre } 0 \leq l \land u < |a| \\
\quad \textbf{post } (\textit{rv} = 1) \leftrightarrow (\exists i.l \leq i \leq u \land a[i] = e) \\
\{ \\
\quad i := l; \\
\quad \textbf{while}(i \leq u) \\
\qquad \{l \leq i \land \forall j.l \leq j < i \rightarrow a[j] \neq e\} \\
\quad \{ \\
\qquad \textbf{if}(a[i] = e) \textbf{ return } 1; \\
\qquad i := i + 1; \\
\quad \} \\
\quad \textbf{return } 0; \}
\end{array}
$$

First basic path:

$$
\begin{array}{l}
\{0 \leq l \land u < |a|\} \\
i := l \\
\{l \leq i \land \forall j.l \leq j < i \rightarrow a[j] \neq e\}
\end{array}
$$

Second basic path:

$$
\begin{array}{l}
\{l \leq i \land \forall j.l \leq j < i \rightarrow a[j] \neq e\} \\
\textbf{while}(i \leq u); \\
\textbf{if}(a[i] = e); \\
\textit{rv} := 1; \\
\{(\textit{rv} = 1) \leftrightarrow (\exists i.l \leq i \leq u \land a[i] = e)\}
\end{array}
$$

assume statement

$\{l \leq i \wedge \forall j. l \leq j < i \rightarrow a[j] \neq e\}$
while$(i \leq u)$;
if$(a[i] = e)$;
rv $:= 1$;
$\{(\mathit{rv} = 1) \leftrightarrow (\exists i. l \leq i \leq u \wedge a[i] = e)\}$

$\{l \leq i \wedge \forall j. l \leq j < i \rightarrow a[j] \neq e\}$
assume $i \leq u$;
assume $a[i] = e$;
rv $:= 1; \{(\mathit{rv} = 1) \leftrightarrow (\exists i. l \leq i \leq u \wedge a[i] =$

Guarded statements introduce **assumptions** about environment

We write these in basic paths using an **assume** statement

assume b means:

1. Rest of path executed only if b is true in current environment
2. When reasoning about rest of path, we can assume b holds

assume: Path Splitting

Each guarded statement introduces two assumptions

One where **assume** b holds, one where **assume** $\neg b$ does

Continuing with our previous basic path, this gives us the next:

$\{l \le i \wedge \forall j.l \le j < i \rightarrow a[j] \ne e\}$
assume $i \le u$;
assume $a[i] = e$;
rv := 1;
$\{(\mathit{rv} = 1) \leftrightarrow (\exists i.l \le i \le u \wedge a[i] = e)\}$

$\{l \le i \wedge \forall j.l \le j < i \rightarrow a[j] \ne e\}$
assume $i \le u$;
assume $a[i] \ne e$;
$i := i + 1$;
$\{l \le i \wedge \forall j.l \le j < i \rightarrow a[j] \ne e\}$

And one final path:

$\{l \le i \wedge \forall j.l \le j < i \rightarrow a[j] \ne e\}$
assume $i > u$;
rv := 0;
$\{(\mathit{rv} = 1) \leftrightarrow (\exists i.l \le i \le u \wedge a[i] = e)\}$

Enumerating Basic Paths

When we enumerate basic paths, we'll follow a convention

Proceed depth-first through the program

When we encounter a guarded command:

1. Assume that it holds first, then generate the ensuing paths
2. Then assume it doesn't hold, proceed as before

Basic Paths: Procedure Calls

Recall the postcondition summarizes the relationships between:

- The procedure's formal parameters
- The return value (special variable *rv*)

We replace procedure calls with postcondition assertions

But postcondition only holds when precondition is satisfied on entry

Introduce another basic path to ensure that the precondition holds

Replace formals in pre/postconditions with actuals appearing in call

Example: Procedure Calls

```
proc BinarySearch(a : array, l, u, e)
  pre 0 ≤ l ∧ u < |a| ∧ sorted(a, l, u)
  post (rv = 1) ↔ (∃i.l ≤ i ≤ u ∧ a[i] = e)
{
  if(l > u) return 0;
  else {
    m := (l + u)/2;
    if(a[m] = e) return 1;
    else if(a[m] < e) {
      return BinarySearch(a, m + 1, u, e);
    } else {
      return BinarySearch(a, l, m − 1, e);
  }
}
```

Example: Procedure Calls

First basic path:

$\{0 \leq l \wedge u < |a| \wedge \text{sorted}(a, l, u)\}$
assume $l > u$;
rv $:= 0$;
$\{(\textit{rv} = 1) \leftrightarrow (\exists i. l \leq i \leq u \wedge a[i] = e)\}$

Second basic path:

$\{0 \leq l \wedge u < |a| \wedge \text{sorted}(a, l, u)\}$
assume $l \leq u$;
$m := (l + u)/2$;
assume $a[m] = e$;
rv $:= 1$;
$\{(\textit{rv} = 1) \leftrightarrow (\exists i. l \leq i \leq u \wedge a[i] = e)\}$

Third basic path:

$\{0 \leq l \wedge u < |a| \wedge \text{sorted}(a, l, u)\}$
assume $l \leq u$;
$m := (l + u)/2$;
assume $a[m] \neq e$;
assume $a[m] < e$;
$\{0 \leq m + 1 \wedge u < |a| \wedge \text{sorted}(a, m + 1, u)\}$

Example: Procedure Calls

Fourth basic path:

$\{0 \leq l \land u < |a| \land \mathsf{sorted}(a, l, u)\}$
assume $l \leq u$;
$m := (l + u)/2$;
assume $a[m] \neq e$;
assume $a[m] < e$;
assume $(v_1 = 1) \leftrightarrow (\exists i. m + 1 \leq i \leq u \land a[i] = e)$;
$\mathit{rv} := v_1$;
$\{(\mathit{rv} = 1) \leftrightarrow (\exists i. l \leq i \leq u \land a[i] = e)\}$

Fifth basic path:

$\{0 \leq l \land u < |a| \land \mathsf{sorted}(a, l, u)\}$
assume $l \leq u$;
$m := (l + u)/2$;
assume $a[m] \neq e$;
assume $a[m] \geq e$;
$\{0 \leq l \land m - 1 < |a| \land \mathsf{sorted}(a, l, m - 1)\}$

Example: Procedure Calls

Sixth basic path:

$$
\begin{array}{l}
\{0 \le l \wedge u < |a| \wedge \mathsf{sorted}(a, l, u)\} \\
\textbf{assume } l \le u; \\
m := (l + u)/2; \\
\textbf{assume } a[m] \ne e; \\
\textbf{assume } a[m] \ge e; \\
\textbf{assume } (v_2 = 1) \leftrightarrow (\exists i. l \le i \le m - 1 \wedge a[i] = e); \\
\mathit{rv} := v_2; \\
\{(\mathit{rv} = 1) \leftrightarrow (\exists i. l \le i \le u \wedge a[i] = e)\}
\end{array}
$$

Summary: Procedure Calls

Given a procedure f with prototype

$$\textbf{proc } \mathsf{f}(x_1, \ldots, x_n)$$
$$\textbf{pre } P[x_1, \ldots, x_n]$$
$$\textbf{post } Q[x_1, \ldots, x_n, \textit{rv}]$$

When f is called in context $w := \mathsf{f}(e_1, \ldots, e_n);$

Augment the calling context with an assertion:

$$\{P[e_1, \ldots, e_n]\};$$
$$w := \mathsf{f}(e_1/x_1, \ldots, e_n/x_n);$$

In paths that pass through the call,

1. Create fresh variable v to hold the return value
2. Replace call with an assumption of the postcondition:

$$\textbf{assume } G[e_1/x_1, \ldots, e_n/x_n, v/\textit{rv}]$$

VC Generation

As we enumerate basic paths, we generate verification conditions

Notice: We only need to generate VCs for three command types

1. Assignment: we do this exactly as we did before
2. Sequence: same as before
3. **assume**. Recall, with assume we said that only paths satisfying the expression proceed past the command.

   $$\text{wlp}(\textbf{assume } b, Q) = b \to Q$$

   If $b \to Q$ holds before, then satisfying b ensures that Q holds afterward

What about the "side conditions" we had for loops?

VC Generation: Loops

$$
\begin{array}{l}
\textbf{proc } \mathsf{f}(x_1, \ldots, x_n) \\
\quad \textbf{pre } P \\
\quad \textbf{post } Q \\
\{ \\
\quad \textbf{while}(b)\ \{I\}\ \{ \\
\qquad c; \\
\quad \} \\
\}
\end{array}
$$

$\{P\}$
skip;
$\{I\}$

$\{I\}$
assume b;
c;
$\{I\}$

$\{I\}$
assume $\neg b$;
$\{Q\}$

VC Generation: Loops

$\{P\}$
skip;
$\{I\}$

$P \rightarrow I$

$\{I\}$
assume b;
c;
$\{I\}$

$I \rightarrow$ wlp(**assume** b, wlp(c, I))
$\Leftrightarrow$
$(I \wedge b) \rightarrow$ wlp(c, I)

$\{I\}$
assume b;
$\{Q\}$

$I \rightarrow$ wlp(**assume** $\neg b, Q$)
$\Leftrightarrow$
$(I \wedge \neg b) \rightarrow Q$

These are same conditions as before!

Example: Linear Search (1)

Recall the first basic path:

$$\begin{array}{l}\{0 \le l \land u < |a|\} \\ i := l \\ \{l \le i \land \forall j. l \le j < i \to a[j] \ne e\}\end{array}$$

The VC for this is:

$$0 \le l \land u < |a| \to \text{wlp}(i := l, l \le i \land \forall j. l \le j < i \to a[j] \ne e)$$

We have that:

$$\begin{array}{rl} & \text{wlp}(i := l, l \le i \land \forall j. l \le j < i \to a[j] \ne e) \\ \Leftrightarrow & l \le l \land \forall j. l \le j < l \to a[j] \ne e \\ \Leftrightarrow & \textit{true} \land \forall j. \textit{false} \to a[j] \ne e \\ \Leftrightarrow & \textit{true}\end{array}$$

Our final condition is valid:

$$0 \le l \land u < |a| \Rightarrow \textit{true}$$

Example: Linear Search (2)

Recall the second basic path:

$$
\begin{aligned}
&\{P : l \leq i \land \forall j. l \leq j < i \rightarrow a[j] \neq e\} \\
&c_1 : \textbf{assume } i \leq u; \\
&c_2 : \textbf{assume } a[i] = e; \\
&c_3 : \mathit{rv} := 1; \\
&\{Q : (\mathit{rv} = 1) \leftrightarrow \exists j. l \leq j \leq u \land a[j] = e\}
\end{aligned}
$$

The VC for this is:

$$P \rightarrow \mathsf{wlp}(c_1; c_2; c_3, Q) \Leftrightarrow P \rightarrow \mathsf{wlp}(c_1, \mathsf{wlp}(c_2, \mathsf{wlp}(c_3, Q)))$$

We have that:

$$
\begin{aligned}
& \mathsf{wlp}(\mathit{rv} := 1, (\mathit{rv} = 1) \leftrightarrow \exists j. l \leq j \leq u \land a[j] = e) \\
\Leftrightarrow \quad & (1 = 1) \leftrightarrow \exists j. l \leq j \leq u \land a[j] = e \\
\Leftrightarrow \quad & \exists j. l \leq j \leq u \land a[j] = e
\end{aligned}
$$

Our final condition is:

$$0 \leq l \land u < |a| \Rightarrow \textit{true}$$

Example: Linear Search (3)

Recall the second basic path:

$$\begin{aligned}
&\{P : l \leq i \wedge \forall j. l \leq j < i \rightarrow a[j] \neq e\} \\
&c_1 : \textbf{assume } i \leq u; \\
&c_2 : \textbf{assume } a[i] = e; \\
&c_3 : \textit{rv} := 1; \\
&\{Q : (\textit{rv} = 1) \leftrightarrow \exists j. l \leq j \leq u \wedge a[j] = e\}
\end{aligned}$$

The VC for this is:

$$P \rightarrow \mathsf{wlp}(c_1; c_2; c_3, Q) \Leftrightarrow P \rightarrow \mathsf{wlp}(c_1, \mathsf{wlp}(c_2, \mathsf{wlp}(c_3, Q)))$$

Moving on,

$$\begin{aligned}
& \mathsf{wlp}(c_1, \mathsf{wlp}(\textbf{assume } a[i] = e, \exists j. l \leq j \leq u \wedge a[j] = e)) \\
\Leftrightarrow\quad & \mathsf{wlp}(c_1, a[i] = e \rightarrow \exists j. l \leq j \leq u \wedge a[j] = e) \\
\Leftrightarrow\quad & \mathsf{wlp}(\textbf{assume } i \leq u, a[i] = e \rightarrow \exists j. l \leq j \leq u \wedge a[j] = e) \\
\Leftrightarrow\quad & i \leq u \rightarrow (a[i] = e \rightarrow \exists j. l \leq j \leq u \wedge a[j] = e)
\end{aligned}$$

Our final condition is:

$$0 \leq l \wedge u < |a| \Rightarrow \textit{true}$$

Example: Linear Search (4)

Our final condition is:

$$
\begin{aligned}
& (l \leq i \leq u \land \forall j. l \leq j < i \rightarrow a[j] \neq e) \rightarrow (a[i] = e \rightarrow \exists j. l \leq j \leq u \land a[j] = e) \\
\Leftrightarrow\ & (l \leq i \leq u \land a[i] = e \land \forall j. l \leq j < i \rightarrow a[j] \neq e) \rightarrow \exists j. l \leq j \leq u \land a[j] = e
\end{aligned}
$$

Notice that:

$$l \leq i \leq u \land a[i] = e \rightarrow \exists j. l \leq j \leq u \land a[j] = e$$

is valid

So, the condition is valid, and the corresponding triple is as well

Next Class

- Mid-term on Thursday
- Check out mid-term guide posted on Blackboard
- Come to office hours with questions