Lecture 25
Verifying Programs with Dynamic Allocation
Instructor: Hossein Hojjat

November 1, 2017
Motivating Example

insert(first, n):
if (first == null)
    first = n;
else {
    n.next = first;
    first.prev = n;
    first = n;
}

How to verify such code?
Motivating Example

insert(first, n):
if (first == null)
   first = n;
else {
   n.next = first;
   first.prev = n;
   first = n;
}

Change of relations (partial functions):
next' = \{ (o_1, o_2), (o_2, o_3), (o_3, o_4), (n, o_1) \} 
prev' = next \cup \{ (o_1, n) \}

using assignments:
next = next[n \mapsto \text{first}]
prev = prev[first \mapsto n]
• Statement
  \[ y = x\text{.next} \]

• Computes the value of \( y \) simply as
  \[ y = \text{next}(x) \]

• We should not de-reference \texttt{null}
  \[
  \texttt{assert}(x \neq \texttt{null});
  y = \text{next}(x)
  \]

• We assume that the type system ensures that if \( x \) is not \texttt{null} then the value \( \text{next}(x) \) is defined

• Otherwise, we could add the corresponding check
  \[
  \texttt{assert}(x \in \text{dom}(\text{next}));
  y = \text{next}(x)
  \]
• We represent each field using a global partial function

• Statement

\[ x.\text{next} = y \]

• Changes heap according to this update:

\[ \text{next}' = \text{next}[x \mapsto y] \]

• which is a notation that expands to:

\[ \text{next}' = \{(u, v) | (u = x \land v = y) \lor (u \neq x \land (u, v) \in \text{next})\} \]

• We should not assign fields of null so we also add this check

\[ \text{assert}(x \neq \text{null}); \]
\[ \text{next}' = \text{next}[x \mapsto y] \]
Why we Need Functions?

- Say we have $x.f$ and $y.f$ in the program
- Why not replace them simply with fresh variables $x_f$ and $y_f$?
- Does this assertion hold for two distinct values $p$, $q$?

```
var xf = ...
var yf = ...
xf = p
yf = q
assert(xf == p)
```

- Yes. The value of $xf$ is still $p$
Why we Need Functions?

- Say we have $x.f$ and $y.f$ in the program
- Why not replace them simply with fresh variables $x_f$ and $y_f$?
- Does this assertion hold for two distinct values $p$, $q$?

```javascript
var xf = ...
var yf = ...
xf = p
yf = q
assert(xf == p)
```

- Yes. The value of $xf$ is still $p$
- Does this assertion hold?

```javascript
...
x.f = p
y.f = q
assert(x.f == p)
```
Why we Need Functions?

- Say we have \( x.f \) and \( y.f \) in the program
- Why not replace them simply with fresh variables \( x_f \) and \( y_f \)?
- Does this assertion hold for two distinct values \( p, q \)?

```
var xf = ...
var yf = ...
xf = p
yf = q
assert(xf == p)
```

- Yes. The value of \( xf \) is still \( p \)
- Does this assertion hold?

```
... 
xf = p
y.f = q
assert(x.f == p)
```

- Depends.
Aliasing

Does the assertion hold in this case:

\[ x = y \]
\[ x.f = p \]
\[ y.f = q \]

**assert** (\(x.f == p\))
Does the assertion hold in this case:

\begin{align*}
x &= y \\
x.f &= p \\
y.f &= q \\
\text{assert}(x.f == p)
\end{align*}

- No! \( y \) and \( x \) are aliased references, denote the same object.
- Even though left hand sides \( x.f \) and \( y.f \) look different, they interfere.
Aliasing

Does the assertion hold in this case:

\[ x = y \]
\[ x.f = p \]
\[ y.f = q \]

\textbf{assert}(x.f == p)

- No! \( y \) and \( x \) are aliased references, denote the same object
- Even though left hand sides \( x.f \) and \( y.f \) look different, they interfere

Does it hold in this case?

\textbf{assume}(x \neq y)
\[ x.f = p \]
\[ y.f = q \]

\textbf{assert}(x.f == p)
Does the assertion hold in this case:

\[
x = y \\
x.f = p \\
y.f = q
\]

\textbf{assert}(x.f == p)

- No! \( y \) and \( x \) are aliased references, denote the same object
- Even though left hand sides \( x.f \) and \( y.f \) look different, they interfere

Does it hold in this case?

\textbf{assume}(x \neq y)

\[
x.f = p \\
y.f = q
\]

\textbf{assert}(x.f == p)

Yes!
Example: wp Computation

- Recall $\text{wp}(x := e, Q) = Q[x \mapsto e]$ (substitution)
- Ignoring null checks, we have the following:

  $$\text{wp}(x.f := p; y.f := q, x.f = p) =$$
  $$\text{wp}(f = f[x \mapsto p]; f = f[y \mapsto q], f(x) = p) =$$
  $$\text{wp}(f = f[x \mapsto p], (f[y \mapsto q])(x) = p) =$$
  $$((f[x \mapsto p])[y \mapsto q])(x) = p$$

- If $h$ is a function then
  $$h[a \mapsto b](u) = v \iff (u = a \land v = b) \lor (u \neq a \land v = h(u))$$
- Thus
  $$((f[x \mapsto p])[y \mapsto q])(x) = p$$
  $$\iff (x = y \land p = q) \lor (x \neq y \land p = (f[x \mapsto p])(x))$$
  $$\iff (x = y \land p = q) \lor (x \neq y \land p = p)$$
  $$\iff (x = y \land p = q) \lor x \neq y$$

Characterizes precisely the weakest condition under which assertion holds.
Exercise

```java
class C {
    var f: C
}
```

- Translate into checks and function updates

\[ x.f.f = z.f + y.f.f.f \]
Exercise

```java
class C {
    var f: C
}
```

- Translate into checks and function updates

\[ x.f.f = z.f + y.f.f.f \]

Solution.

```plaintext
assume(z \neq null)
assume(y \neq null)
assume(f(y) \neq null)
assume(f(f(y)) \neq null)
assume(f(x) \neq null)

f := f \left[ f(x) \mapsto (f(z) + f(f(f(y)))) \right]
```
• Can we prove this?

```java
x = new C();
y = new C();
assert (x != y);
```
Modeling Dynamic Allocation

- Can we prove this?

```java
x = new C();
y = new C();
assert(x != y);
```

- Can we introduce global variables and assumptions that correctly describe fresh objects?
• Can we prove this?

```java
x = new C();
y = new C();
assert(x != y);
```

• Can we introduce global variables and assumptions that correctly describe fresh objects?

• Global set `alloc` denotes objects allocated so far

```java
x = new C();
```

• denotes (for now):

```java
havoc(x);
assume(x \notin alloc);
alloc = alloc \cup \{x\}
```
Original program

\[ x = \text{new } C(); \]
\[ y = \text{new } C(); \]
\[ \text{assert}(x \neq y); \]

Becomes

\[ \text{havoc}(x); \]
\[ \text{assume}(x \notin \text{alloc}) \]
\[ \text{alloc} = \text{alloc} \cup \{x\}; \]
\[ \text{havoc}(y); \]
\[ \text{assume}(y \notin \text{alloc}); \]
\[ \text{alloc} = \text{alloc} \cup \{y\}; \]
\[ \text{assert}(x \neq y); \]

Renaming variables we obtain:

\[ \text{havoc}(x); \]
\[ \text{assume}(x \notin \text{alloc}) \]
\[ \text{alloc}_1 = \text{alloc} \cup \{x\}; \]
\[ \text{havoc}(y); \]
\[ \text{assume}(y \notin \text{alloc}_1); \]
\[ \text{alloc}_2 = \text{alloc}_1 \cup \{y\}; \]
\[ \text{assert}(x \neq y); \]

Assertion holds because

\[
(\text{alloc}_1 = \text{alloc} \cup \{x\}) \land \\
(y \notin \text{alloc}_1) \Rightarrow \quad x \neq y
\]