Synthesis, Analysis, and Verification

Lecture 05b

Dynamic Allocation

Linked Structures and Their Properties

WS1S

Lectures:

Viktor Kuncak

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Today we talk about something **new**

Memory Allocation in Java

{true} (Hoare triple)

x = **new** C();

y = **new** C();

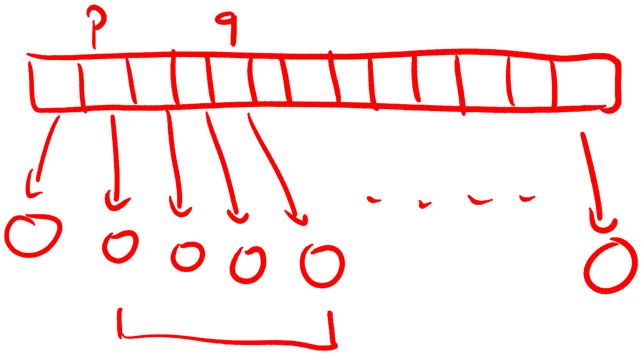
assert(x != y); // fresh object references-distinct

Why should this assertion hold?

How to give meaning to ‘new’ so we can prove it?

How to represent fresh objects?

```
assume(N > 0 && p > 0 && q > 0 && p != q);
a = new Object[N];
i = 0;
while (i < N) {
  a[i] = new Object();
  i = i + 1;
}
assert(a[p] != a[q]);
```



A View of the World

Everything exists, and will always exist.
(It is just waiting for its time to become allocated.)

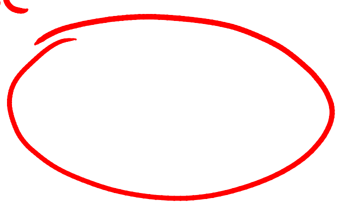
It will never die (but may become unreachable).

alloc : Obj → Boolean   i.e.   alloc : Set[Obj]

x = new C();   →   havoc(x);
assume($x \notin alloc$);
$alloc_1 = alloc \cup \{x\}$; $x \in alloc_1$

^defult constructor

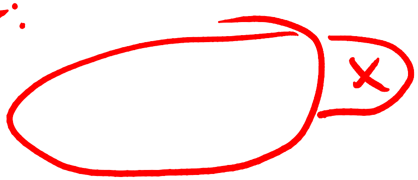
before:
alloc



y = new C()
assert($x \neq y$)

havoc(y);
assume($y \notin alloc_1$); $y \notin alloc_1$
$alloc_2 = alloc_1 \cup \{y\}$

after:



New Objects Point Nowhere

class C { int f; C next; C prev; }

this should work:

```
x = new C();
assert(x.f==0 && c.next==null  && c.prev==null)
```

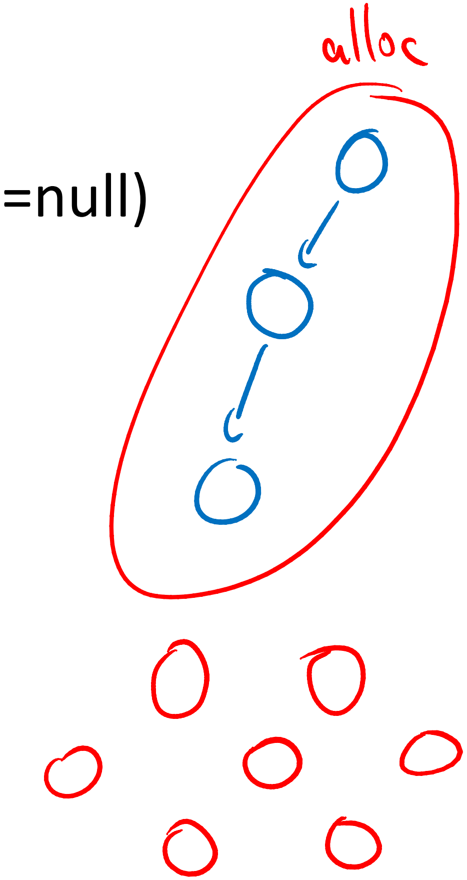
x = new C(); →

havoc(x)
assume(x ∉ alloc)
alloc = alloc ∪ {x}

assume (f(x)==0 ∧
next(x)=null ∧
prev(x)=null);

1) use assignment
f= f(x:=0)

2) use assume



If you are new, you are known by few

class C { int f; C next; C prev; }

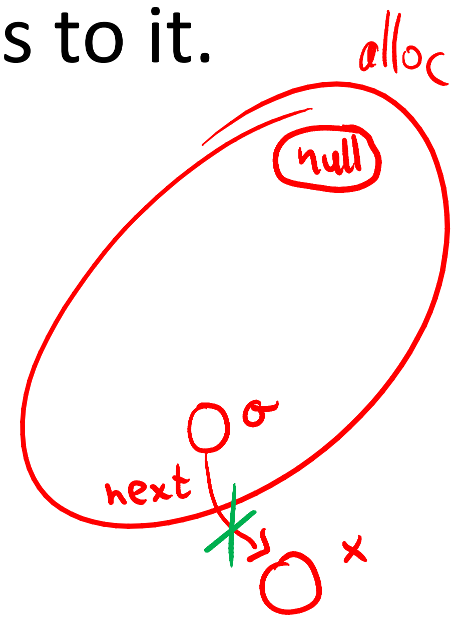
Assume C is the only class in the program

Lonely object: no other object points to it.

Newly allocated objects are lonely!

x = new C(); → [

$\forall \sigma, \sigma \in alloc \rightarrow next(\sigma) \neq x$



$\forall \sigma. \quad \sigma \in alloc \rightarrow next(\sigma) \in alloc \wedge prev(\sigma) \in alloc$

Remember our Model of Java Arrays

```
class Array {
  int length;
  data : int[]
}
```

length : Array -> int

data : Array -> (Int -> Int)

or simply: Array x Int -> Int

a[i] = x ➔

assert $(a \neq null)$;
assert $(0 \leq i \wedge i < length(a))$;
data= data((a,i):= x)

y = a[i] ➔

assert $(a \neq null)$;
assert $(0 \leq i \wedge i < length(a))$
$y = data((a, i))$

Allocating New Array of Objects

class oArray {
  int length;
  data : Object[]
}

x = new oArray[100] →

E

havoc (x);
assume ($x \notin alloc$);
alloc = alloc $\cup \{x\}$;
assume (length (x) = E (100) $\wedge$
$\forall i.\ 0 \le i < E \rightarrow$ data (x, i) = null $\wedge$
$\forall o \in alloc.\ \bigwedge_{f \in fields(C(o))} f(o) \neq x$

length = length (x := E)

Procedure Contracts

Suppose there are fields and variables f_1,f_2,f_3 (denoted f)

procedure foo(x):

requires P(x,f)

modifies f_3

ensures Q(x,old(f),f)

foo(E) →

assert(P(E,f));

old_f = f;

havoc(f_3);

assume Q(E,old_f, f)

Modification of Objects

Suppose there are fields and variables $\underbrace{f_1,f_2,f_3}_{f}$ (denoted f)

procedure foo(x):

requires P(x,f)

modifies x.f_3

ensures Q(x,f,f')

$x.f_3 = y \rightsquigarrow f_3 = f_3(x := y)$

foo(E) →

assert(P(E,f));

old_f = f; ← $\begin{cases} old\_f_1 = f_1 \\ old\_f_3 = f_3 \end{cases}$

havoc(x.f_3); → havoc(f_3); assume $\forall z \neq x.\ f_3(z) = old\_f_3(z)$

assume Q(E,old_f, f)

Example

```
class Pair { Object first; Object second; }
void printPair(p : Pair) { ... }
void printBoth(x : Object, y : Object)
modifies first, second  // ?
{
  Pair p = new Pair();
  p.first = x;
  p.second = y;
  printPair(p);
}
```

printBoth (x1,y1)

Allowing Modification of Fresh Objects

Suppose there are fields and variables f_1, f_2, f_3 (denoted f)

procedure foo(x):
  requires P(x,f)
  modifies $x.f_3$
  ensures Q(x,f,f')

foo(E) →
  assert(P(E,f));
  old_f = f; old_alloc = alloc;
  havoc f_3, f_2, f_1, alloc
  assume $\forall z \in \text{old\_alloc}.\ f_1(z) = \text{old\_}f_1(z) \wedge f_2(z) = \text{old\_}f_2(z)$
  $(z \neq x \rightarrow f_3(z) = \text{old\_}f_3(z))$
  assume Q(E,old_f, f)
  assume (old_alloc ⊆ alloc)

Data remains same if: 1) existed and 2) not listed in m.clause

Quiz will be this Tuesday!
(not open book)
Bring: paper, pen, EPFL Camipro card

Now we can model many programs

We can represent any body of sequential code inside one procedure.

Our loop invariants, pre/post conditions can become very complex

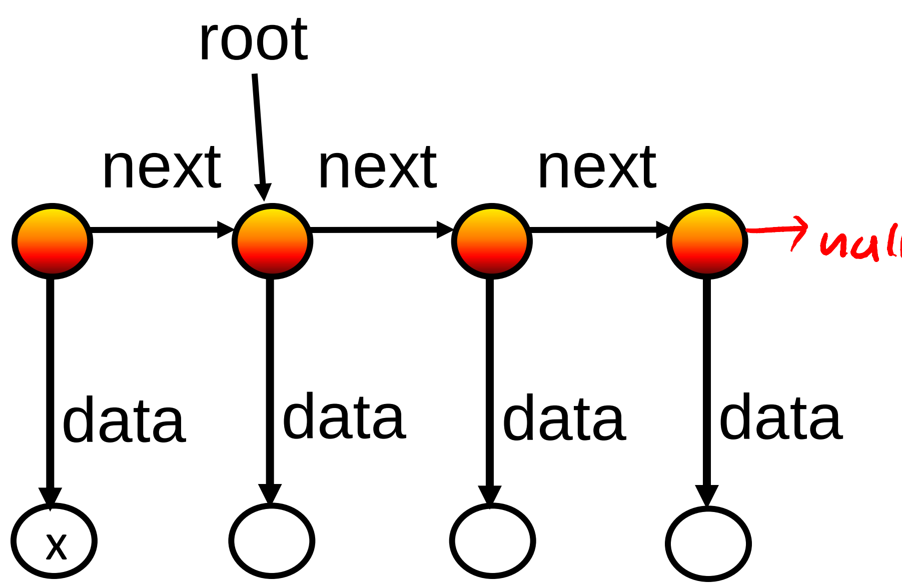
Linked List Implementation

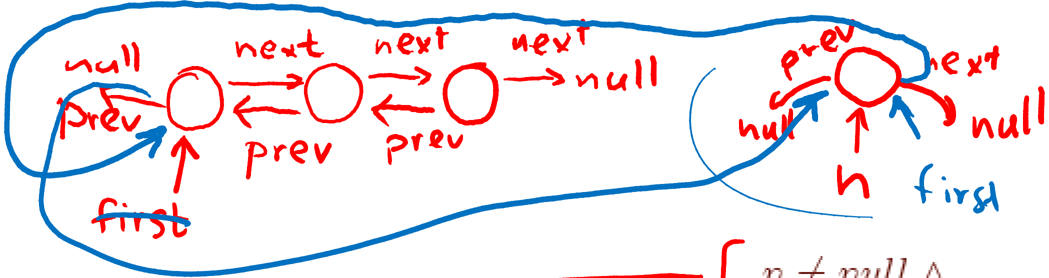
```
class List {
  private List next;
  private Object data;
  private static List root;
```

content = "objects reachable from 'root'" = $\{root\} \bullet next^{*} \bullet data$

content = $old(content) \cup \{x\}$

```
  public static void addNew(Object x) {
     List n1 = new List();
     n1.next = root;
     n1.data = x;
     root = n1;
  }
}
```





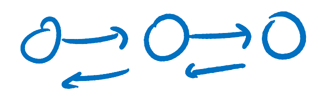
Doubly Linked

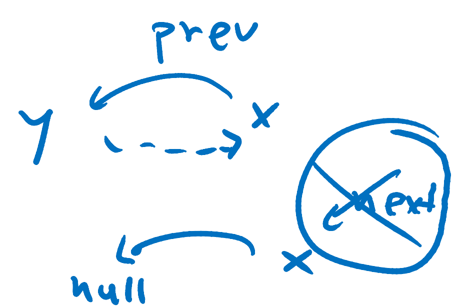
```
assume P;
if (first == null) {
  first = n;
  n.next = null;
  n.prev = null;
} else {
  n.next = first;
  first.prev = n;
  n.prev = null;
  first = n;
}
assert Q;
```

$$n \neq null \wedge next(n) = null \wedge prev(n) = null \wedge prev(first) = null \wedge Q$$

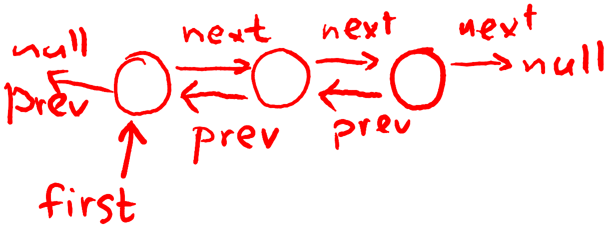
prev(null)=null
next(null)=null

assume(first != null)
assert(n ≠ null);
next = next(n:=first)
prev = prev(first:=n)
prev = prev(n:=null)
first = n





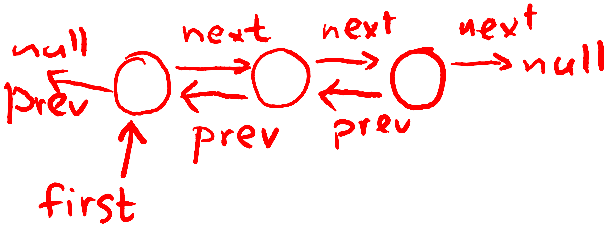
$$\forall x.\forall y.prev(x) = y \rightarrow (y \neq null \rightarrow next(y) = x) \wedge (y = null \wedge x \neq null \rightarrow (\forall z.next(z) \neq x))$$




```
assume P;
if (first == null) {
  first = n;
  n.next = null;
  n.prev = null;
} else {
  n.next = first;
  first.prev = n;
  n.prev = null;
  first = n;
}
assert Q;
```

$$\begin{cases} n \neq null \land \\ next(n) = null \land prev(n) = null \land \\ prev(first) = null \ \land \ Q \end{cases}$$

$$\begin{cases} \forall x. \forall y. prev(x) = y \quad \rightarrow \\ \quad (y \neq null \rightarrow next(y) = x) \land \\ \quad (y = null \land x \neq null \rightarrow (\forall z. next(z) \neq x)) \end{cases}$$



Reachability

```
assume P;
if (first == null) {
  first = n;
  n.next = null;
  n.prev = null;
} else {
  n.next = first;
  first.prev = n;
  n.prev = null;
  first = n;
}
assert Q;
```

$$\left\{\begin{array}{l} n \neq null \wedge \\ next(n) = null \wedge prev(n) = null \wedge \\ prev(first) = null \ \wedge \ Q \end{array}\right.$$

$$\left\{\begin{array}{l} \forall x.\forall y.prev(x) = y \quad \rightarrow \\ \qquad (y \neq null \rightarrow next(y) = x) \wedge \\ \qquad (y = null \wedge x \neq null \rightarrow (\forall z.next(z) \neq x)) \end{array}\right.$$

How to prove such verification conditions automatically?