Verifying Programs with Arrays

Computer Science and Artificial Intelligence Laboratory
MIT

Oct 28, 2015

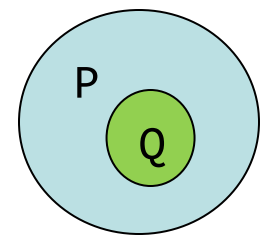
Recap: Weakest Preconditions

$$\mathrm{P} = wpc(c, A)$$

Command — Predicate

Weakest predicate P such that $\vDash \{P\}\, c\, \{A\}$

- P weaker than Q iff $Q \Rightarrow P$



$$\text{wpc}(\text{skip } \{Q\}) = Q$$

$$\text{wpc}(x = e\{Q\}) = Q[e/x]$$

$$\text{wpc}(C1; C2\{Q\}) = \text{wpc}(C1\{\text{wpc}(C2\{Q\})\})$$

$$\text{wpc}(\text{if } B \text{ then } C1 \text{ else } C2\{Q\}) =$$
$$(B \text{ and } \text{wpc}(C1\{Q\})) \text{ or } (\text{not } B \text{ and } \text{wpc}(C2\{Q\}))$$

Recap: Weakest Precondition

While-loop is tricky

- Let W $= wpc(while\ e\ do\ c, B)$
- then,

$$\mathrm{W} = e \Rightarrow wpc(c, W) \ \land \ \neg e \Rightarrow B$$

Recap: Verification Condition

Stronger than the weakest precondition

Can be computed by using an invariant

$$VC(while_I\ e\ do\ c, B) =$$
$$I \wedge\ \forall x_1, \ldots x_n\ I \Rightarrow (e \Rightarrow VC(c, I) \wedge \neg e \Rightarrow B)$$

- Where x_i are variables modified in c.

The problem with arrays

```
{true}
a[k]=1;
a[j]=2;
x=a[k]+a[j];
{x=3}
```

```
{true}
a[k]=1;
a[j]=2;
{a[k]+a[j]=3}
x=a[k]+a[j];
{x=3}
```

Now what?

Can we use the standard rule for assignment?

$$wpc(x := e, C) \ = \ C[x \rightarrow e]$$

The problem with arrays

```
{true}
a[k]=1;
a[j]=2;
x=a[k]+a[j];
{x=3}
```

```
{true}
a[k]=1;
a[j]=2;
{a[k]+a[j]=3}
x=a[k]+a[j];
{x=3}
```

```
{true}
{1+2=3}
a[k]=1;
{a[k]+2=3}
a[j]=2;
{a[k]+a[j]=3}
x=a[k]+a[j];
{x=3}
```

What if k=j?

Theory of arrays

Let a be an array

$a\{i \to e\}$ is a new array whose ith entry has value e

- $a\{i \to e\}[k] = \begin{cases} a[k] & if\ k \neq i \\ e & if\ k = i \end{cases}$

A formula involving TOA can be expanded into a set of implications.

- Ex. Assume Zero is the zeroed out array

- $Zero\{i \to 5\}\{j \to 7\}[k] = 5 \iff \quad \ldots$

Assignment rule with theory of arrays

$$\frac{}{\vdash \{P[a \to a\{i \to e\}]\}\;\; a[i] = e \;\; \{P\}}$$

```
{true}
a[k]=1;
a[j]=2;
{a[k]+a[j]=3}
x=a[k]+a[j];
{x=3}
```

→

```
{true}
{a{k->1}{j->2}[k]+a{k->1}{j->2}[j]=3}
a[k]=1;
{a{j->2}[k]+a{j->2}[j]=3}
a[j]=2;
{a[k]+a[j]=3}
x=a[k]+a[j];
{x=3}
```

Assignment rule with theory of arrays

$$\frac{}{\vdash \{P[a \ \to a\{i \to e\}]\} \ \ a[i] = e \ \ \{P\}}$$

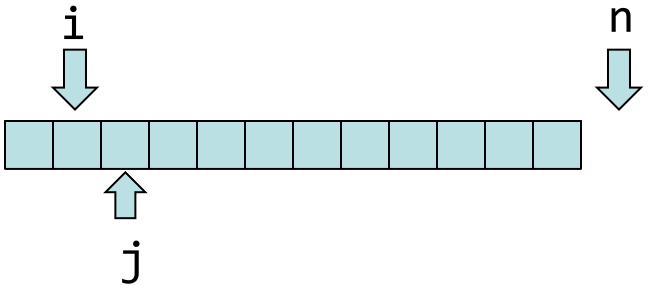
```
{true}
a[k]=1;
a[j]=2;
{a[k]+a[j]=3}
x=a[k]+a[j];
{x=3}
```

$\{k \neq j\}$

```
{a{k->1}{j->2}[k]+a{k->1}{j->2}[j]=3}
a[k]=1;
{a{j->2}[k]+a{j->2}[j]=3}
a[j]=2;
{a[k]+a[j]=3}
x=a[k]+a[j];
{x=3}
```

Arrays and loops

Consider the following program:



$\{0 \leq i < n\}$

```
j = i+1
while (j<n) {
    a[i] = max(a[i],a[j]);
    j = j+1;
}
```

$\{\forall_{i \leq k < n} a_0[k] \leq a[i]\}$

A reasonable loop invariant: $\forall_{i \leq k < j} a_0[k] \leq a[i]$

Initial array

Arrays and loops

Let's try to verify our candidate loop invariant

$\{\forall_{i \le k < j} a_0[k] \le a[i]\}$

$\{\forall_{i \le k < j+1} a_0[k] \le \max(a[i], a[j])\}$

$\{\forall_{i \le k < j+1} a_0[k] \le a[i \to \max(a[i], a[j])][i]\}$

```
a[i] = max(a[i],a[j]);
```

$\{\forall_{i \le k < j+1} a_0[k] \le a[i]\}$

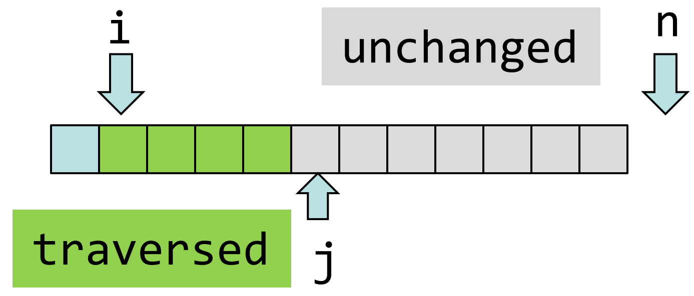
```
j = j+1;
```

$\{\forall_{i \le k < j} a_0[k] \le a[i]\}$

We can't quite prove this implication!

We don't know that $a_0[j] \le a[j]$

A better loop invariant



$\{\forall_{i \le k < j} a_0[k] \le a[i] \land \forall_{j \le k < n} a_0[k] = a[k]\}$

$\{\forall_{i \le k < j+1} a_0[k] \le \max(a[i], a[j]) \land \forall_{j+1 \le k < n} a_0[k] = k = i ? \ldots : a[k]\}$

$\{\forall_{i \le k < j+1} a_0[k] \le a[i \to \max(a[i], a[j])][i]$
$\land \forall_{j+1 \le k < n} a_0[k] = a[i \to \max(a[i], a[j])][k]\}$

```
a[i] = max(a[i],a[j]);
```

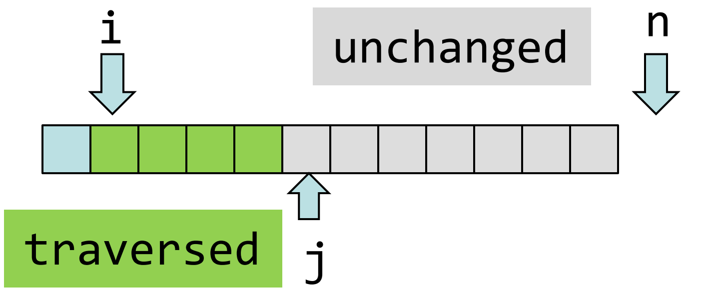
$\{\forall_{i \le k < j+1} a_0[k] \le a[i] \land \forall_{j+1 \le k < n} a_0[k] = a[k]\}$

```
j = j+1;
```

$\{\forall_{i \le k < j} a_0[k] \le a[i] \land \forall_{j \le k < n} a_0[k] = a[k]\}$

We don't know that $k \neq i$

An even better invariant



$\{\forall_{i \le k < j} a_0[k] \le a[i] \wedge \forall_{j \le k < n} a_0[k] = a[k] \wedge i < j\}$

$\{\forall_{i \le k < j+1} a_0[k] \le \max(a[i], a[j]) \wedge \forall_{j+1 \le k < n} a_0[k] = k = i ? \ldots : a[k] \wedge i < j\}$

$\{\forall_{i \le k < j+1} a_0[k] \le a[i \to \max(a[i], a[j])][i]$
$\quad \wedge \forall_{j+1 \le k < n} a_0[k] = a[i \to \max(a[i], a[j])][k] \wedge i < j + 1\}$

```
a[i] = max(a[i],a[j]);
```

$\{\forall_{i \le k < j+1} a_0[k] \le a[i] \wedge \forall_{j+1 \le k < n} a_0[k] = a[k] \wedge i < j + 1\}$

```
j = j+1;
```

$\{\forall_{i \le k < j} a_0[k] \le a[i] \wedge \forall_{j \le k < n} a_0[k] = a[k] \wedge i < j\}$

Arrays in SMT-LIB

New constructs you need to know:

```
(define-sort A () (Array Int Int))
(declare-fun a1 () A)

(select a1 x)
(store a1 x y)

(forall ((k Int)) ...)
```

Example

Encode that the invariant from before is preserved by the loop body

$$I \wedge cond \Rightarrow VC(body, I)$$

$$\left(\forall_{i \le k < j} a_0[k] \le a[i] \wedge \forall_{j \le k < n} a_0[k] = a[k] \wedge i < j\right) \wedge (j < n) \Rightarrow \forall_{i \le k < j+1} a_0[k] \le a[i \to \max(a[i], a[j])][i] \wedge \forall_{j \le k < n} a_0[k] = a[i \to \max(a[i], a[j])][k] \wedge i < j + 1$$

Useful links

Z3 web interface and examples:

http://rise4fun.com/Z3

Z3 tutorial:

http://rise4fun.com/z3/tutorial

6.820 Fundamentals of Program Analysis
Fall 2015