Some data-flow based techniques for assertion checking in concurrent programs

Deepak D'Souza

Department of Computer Science and Automation
Indian Institute of Science, Bangalore.

[with inputs from Suvam Mukherjee and Nisant Sinha]

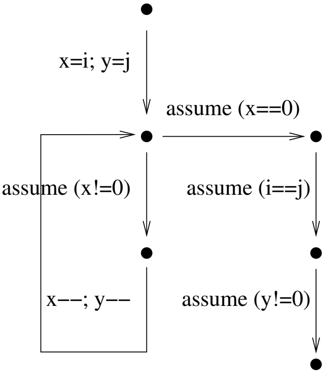
27 July 2013

Outline

1. Inductive annotations for programs
2. Data-Flow analysis
3. Data-Flow graphs

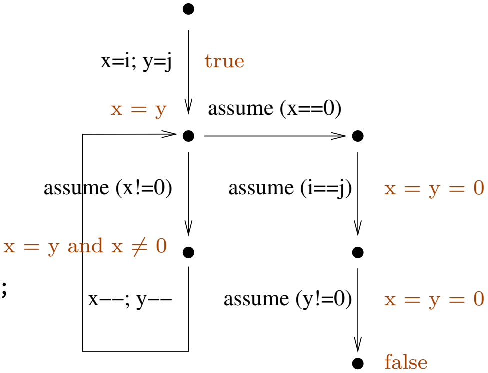
Inductive annotation as proof for sequential programs

```
x = i; y = j;
while (x!=0){
   x--; y--;
}
if (i==j)
   assert (y=0);
```



Inductive annotation as proof for sequential programs

```
x = i; y = j;
while (x!=0){
   x--; y--;
}
if (i==j)
   assert (y=0);
```



What about concurrent programs?

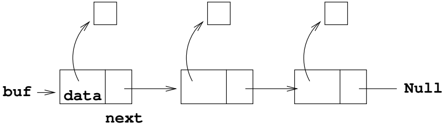
- Collecting state for a concurrent program.

$$x := 0$$

$$\begin{array}{lcl} i := 0 & & x := 2 \\ x := 1 & \| & j := 1 \\ y := x + 1 & & \end{array}$$

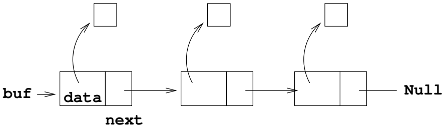
Example analysis using Radar [CVJL2008]

Adjusted Analysis	buffer_list *bufs; lock buf_lock; int perf_ctr;
	thread producer1(){
∅	P0: px = bufs;
∅	P1: while (px != NULL){
px	P2: lock(buf_lock);
px	P3: px->data = new();
px->data, px	P5: perf_ctr++;
px->data, px	P6: t=produce();
px->data, px	P8: *px->data = t;
px->data, px	P9: unlock(buf_lock);
~~px->data~~, px	PA: px = px->next;
	}
	thread consumer1(){
∅	perf_ctr = 0;
∅	C0: cx = bufs;
∅	C1: while(cx != NULL){
cx	C2: lock(buf_lock);
cx	C3: if(cx->data != NULL){
cx->data, cx	C4: consume(*cx->data);
cx->data, cx	C5: cx->data = NULL;
cx	C6: cx = cx->next;
	}
∅	C7: unlock(buf_lock);
	}



Example analysis using Radar: Program 2

Adjusted Analysis	
	`buffer_list *bufs;`
	`lock buf_lock;`
	`int perf_ctr;`
	`thread producer1(){`
$\emptyset$	`P0: px = bufs;`
$\emptyset$	`P1: while (px != NULL){`
px	`P2:   lock(buf_lock);`
px	`P3:    px->data = new();`
px->data, px	`P4:    unlock(buf_lock);`
~~px->data~~, px	`P5:    perf_ctr++;`
~~px->data~~, px	`P6:    t=produce();`
~~px->data~~, px	`P7:    lock(buf_lock);`
~~px->data~~, px	`P8:    *px->data = t;`
~~px->data~~, px	`P9:    unlock(buf_lock);`
~~px->data~~, px	`PA:    px = px->next;`
	`    }`
	`thread consumer1(){`
$\emptyset$	`    perf_ctr = 0;`
$\emptyset$	`C0: cx = bufs;`
$\emptyset$	`C1: while(cx != NULL){`
cx	`C2:   lock(buf_lock);`
cx	`C3:    if(cx->data != NULL){`
cx->data, cx	`C4:      consume(*cx->data);`
cx->data, cx	`C5:      cx->data = NULL;`
cx	`C6:      cx = cx->next;`
	`      }`
$\emptyset$	`C7:    unlock(buf_lock);`
	`    }`



Data-Flow analysis for datarace-free programs [De et al 2011]

- Lifts any value-set analysis (like constant progagation, non-null analysis) to a sound analysis for concurrent programs.
- Is potentially precise at non-racy reads.
- Add "sync" edges between statements across threads (from `unlock` to `lock`).
- Find the least fix point over this "sync-CFG".

Data-Flow analysis for datarace-free programs [De et al 2011]

```
                   buf* p;
                   lock l;
                   main() {
                   M1: p = new(...);
                p  M2: p->data = new(...);
        p,p->data  M3: *p->data = DEFAULTVAL;
        p,p->data  M4: spawn("prod");
        p,p->data  M5: spawn("cons");
                         ...
                   }
```

```
               prod() {
               P1: while(1) {
                     ...
    p,p->data  P2:   lock(l);
    p,p->data  P3:   oldv = *p->data;
            p  P4:   p->data = NULL;
            p  P5:   newv = nextv(oldv);
            p  P6:   p->data = new(...);
    p,p->data  P7:   *p->data = newv;
    p,p->data  P8:   unlock(l);
                     ...
                   }
               }
```

```
cons() {                      p,p->data
C1: while(1) {
      ...
C2:   lock(l);                p,p->data
C3:   v = *p->data;           p,p->data
C4:   unlock(l);              p,p->data
      ...
    }
}
```

Modular reasoning about data and control [Farzan,Kincaid 2012]

- Define a notion of inductive annotation based on data-flow graphs.
- Show how to compute such a graph
  - Begin with intra-thread data-flow edges
  - Compute an inductive annotation for this DFG
  - Check if any more edges need to be added. If so repeat, else done.

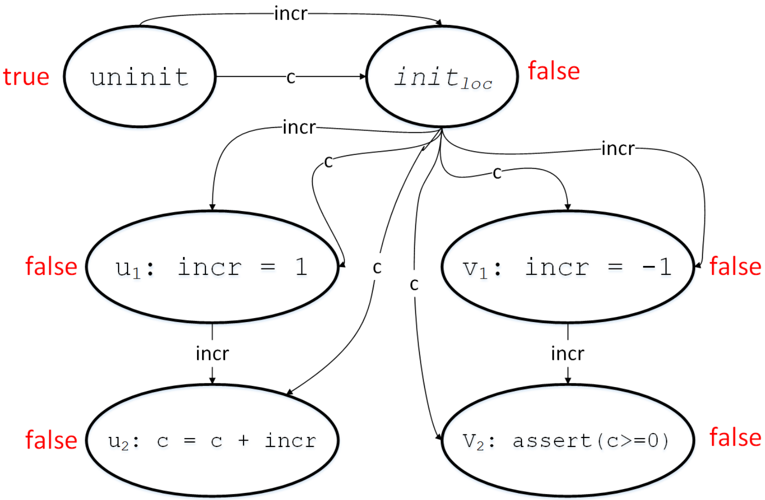
x := 0	||	lock(l)
lock(l)		read x
x := −1		unlock(l)
x := 0		
unlock(l)		

Example program from [FK2012]

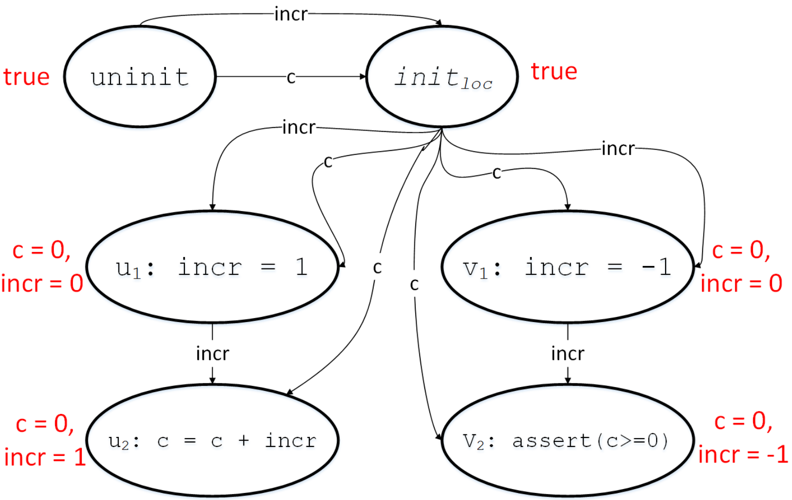
$$
\begin{array}{lcl}
c := 0 & & c := 0 \\
incr := 0 & \Big|\Big| & incr := 0 \\
incr := 1 & & incr := -1 \\
c := c + incr & & \text{assert } (c \geq 0)
\end{array}
$$

- Multiple instances of 2 static threads.
- Variable `incr` is local to each thread.

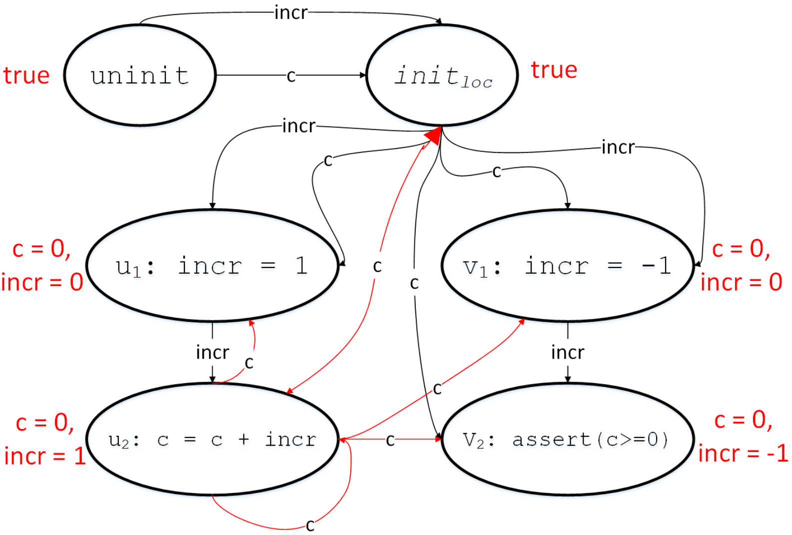
Example: Initial DFG using intra-thread data-flow edges



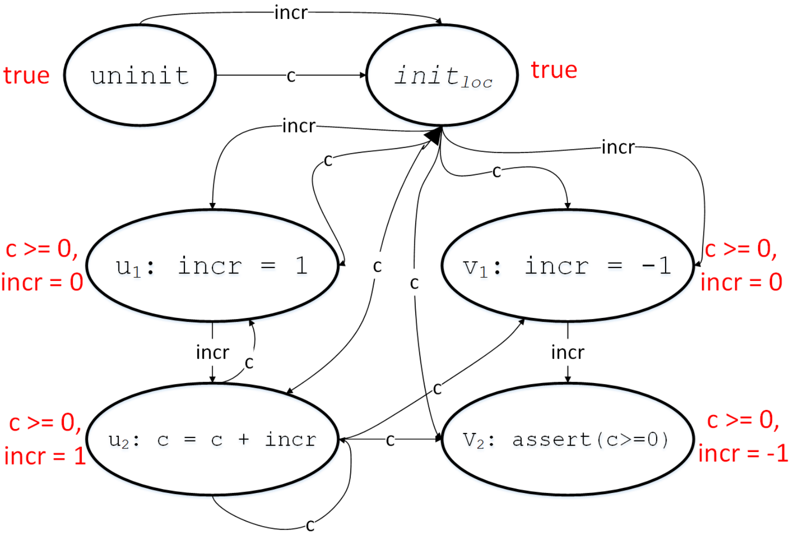
Example: Initial DFG with inductive annotation ι



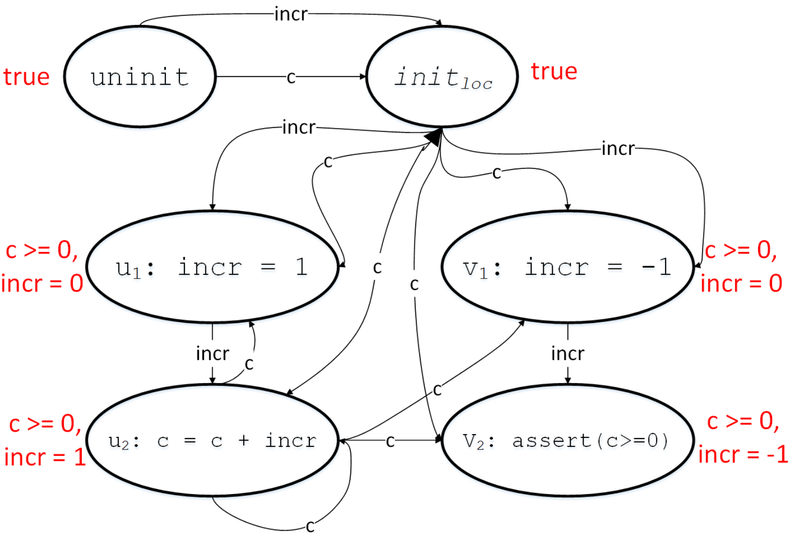
Example: Control-flow analysis: Adding ι-feasible data-flow edges



Example: Iteration 2: Compute new inductive annotation ι_2



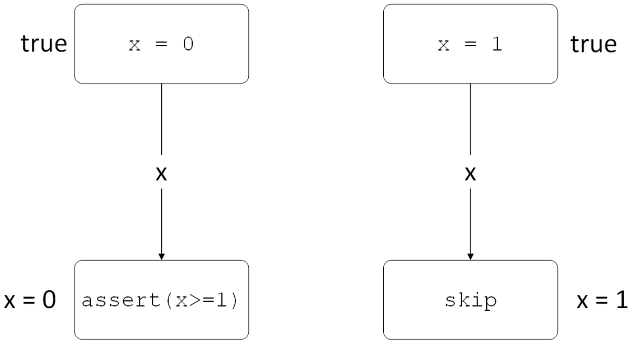
Example: Iteration 2: Check if any ι_2-feasible data-flow edges can be added



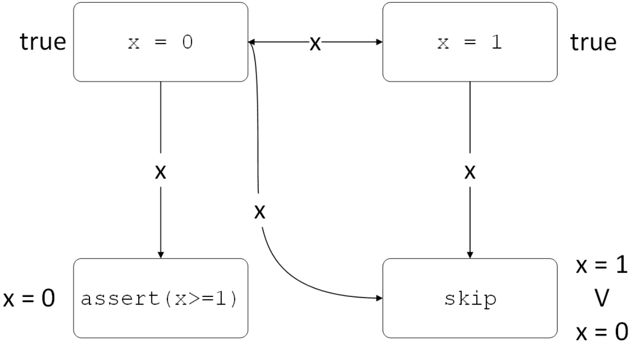
Why ι-infeasibility helps

x := 0	$\|\|$	lock(l)
lock(l)		read x
x := −1		unlock(l)
x := 0		
unlock(l)		

Problematic example



Problematic example



Problematic example