Preventing Races and Deadlocks in Concurrent Programs: The SHIM Approach

Nalini Vasudevan and Stephen A. Edwards
Columbia University

Data Races

```
                    int x = 1;
void foo() {                          void bar() {
        spawn(bar);                           lock(M);
        lock(M);                              x = x*2;
        x++;                                  unlock(M);
        unlock(M);                    }
        print(x);
        // Prints 3 or 4
}
```

Deadlocks

```
                    int x = 1;
void foo() {                          void bar() {
        spawn(bar);                           lock(M);
        lock(N);                              lock(N);
        lock(M)                               x = x*2;
        x++;                                  unlock(N)
        unlock(M);                            unlock(M);
        unlock(N);                    }
}
```

The SHIM Programming Language

- Stands for *Software Hardware Integration Medium*
- Race free, scheduling independent
- Blocking synchronous rendezvous communication

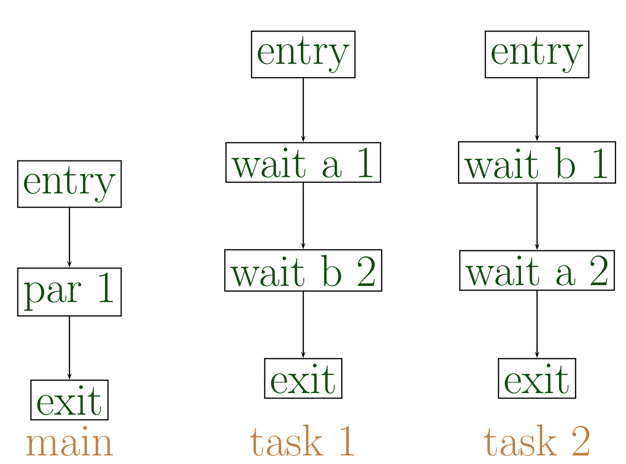
stmt$_1$ *par* *stmt*$_2$	Run *stmt*$_1$ and *stmt*$_2$ concurrently
send var	Send on channel *var*
recv var	Receive on channel *var*

```
void main() {
 chan int a, b;
 { // Task 1
    a = 15, b = 10;
    send a;
    send b;
 } par { // Task 2
    int c;
    recv a;
    recv b;
    c = a + b;
    // value of c is 25
  }
}
```

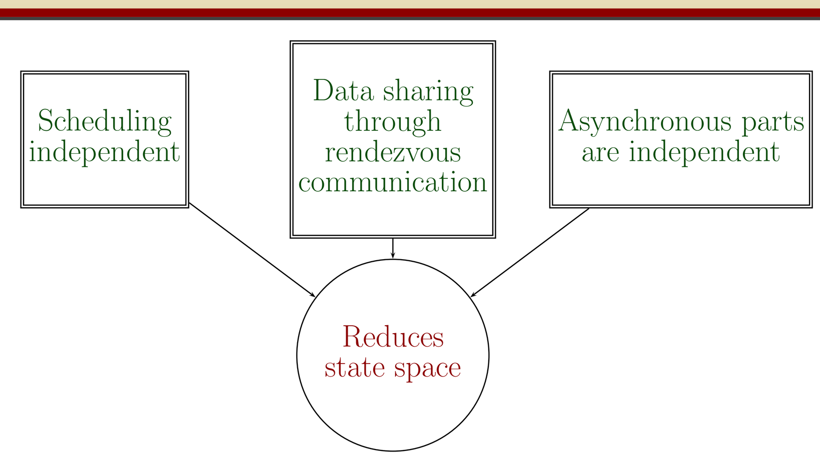
```
void f(chan int a) {    // a is a copy of c
  a = 3;                // change local copy
  recv a;               // receive (wait for g)
  // a now 5
}
void g(chan int &b) {   // b is an alias of c
   b = 5;               // sets c
   send b;              // send (wait for f)
  // b now 5
}
void main() {
  chan int c = 0;
  f(c); par g(c);
  c = c * 2;    // c is now 10
}
```

The Problem with SHIM: Deadlock

```
void main() {
 chan int a, b;
 {
    // Task 1
    a = 15, b = 10;
    send a;
    send b;
 } par {
    // Task 2
    int c;
    recv b;
    recv a;
    c = a + b;
  }
}
```



SHIM Design for Static Deadlock Detection



Abstraction and Modeling

```
ready_a :=
  main = par_1 &
  task_1 =  wait_a_1 &
  task_2 =  wait_a_2
```

Rendezvous Condition

```
next(task_1) :=
 case
  (task_1 = entry) & (main = par_1): wait_a_1;
  (task_1 = wait_a_1) & ready_a: wait_b_2;
  (task_1 = wait_b_2) & ready_b: exit;
  1: task_1;
 esac;
```

Transitions for task 1

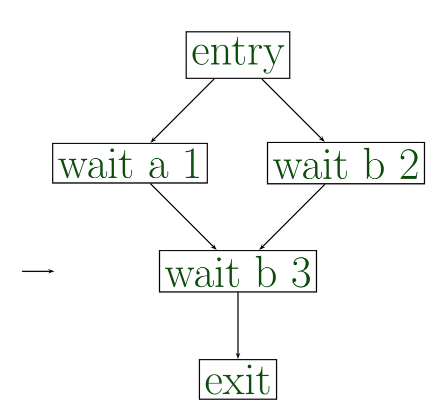
Checking for absence of deadlock:

- Maintain a progress bit for each task
- If no task progresses, then the program is in the deadlock state

```
SPEC AG((main!= exit) -> ( progress_main = yes
       | progress_task_1 = yes | progress_task_2 = yes))
```

Conditional Statements

```
if (n)
   send a;
else
   recv b;
recv b;
```




```
(task_1 = entry) &
(main = par_1):
   {wait_a_1, wait_b_2};
```

- We abstract the condition
- May lead to false positives

Results

Example	Lines	Channels	Tasks	Result	Runtime	Memory
Source-Sink	35	2	11	No Deadlock	0.2 s	3.9 MB
Pipeline	30	7	13	No Deadlock	0.1	2.0
Prime Sieve	35	51	45	No Deadlock	1.7	25.4
Berkeley	40	3	11	No Deadlock	0.2	7.2
FIR Filter	100	28	28	No Deadlock	0.4	13.4
Bitonic Sort	130	65	167	No Deadlock	8.5	63.8
Framebuffer	220	11	12	No Deadlock	1.7	11.6
JPEG Decoder	1020	7	15	May Deadlock	0.9	85.6
JPEG Modified	1025	7	15	No Deadlock	0.9	85.6

Conclusions

- SHIM: A deterministic concurrent model
- We can statically detect deadlocks
  - Using synchronous methodologies to verify asynchronous systems.
- Future Work
  - Increase channel buffer size to increase performance and avoid deadlocks
  - Convince the world: **SHIM's philosophy- Deadlocks are better than data races**