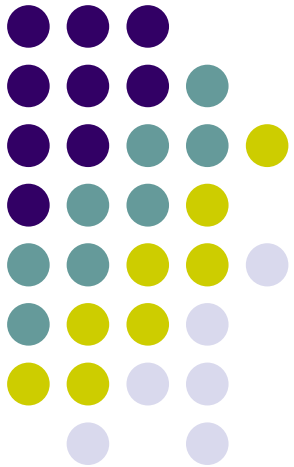
Choosing Among Alternative Futures

Steve MacDonald[1], Jun Chen[1]
and Diego Novillo[2]

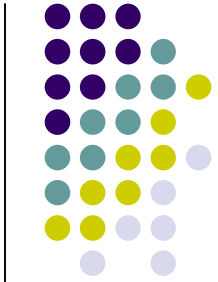
University of Waterloo[1], Canada
Red Hat Canada[2]



University of Waterloo

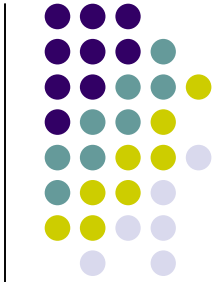
redhat.

Outline



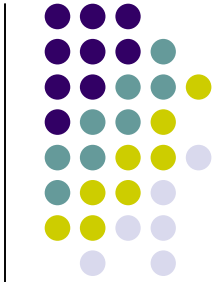
- Motivation
- Static Analysis
  - SSA, CSSA, CSSAME
- Aspect Technology
- Implementation
- Limitations and Future Work
- Conclusion

Motivation

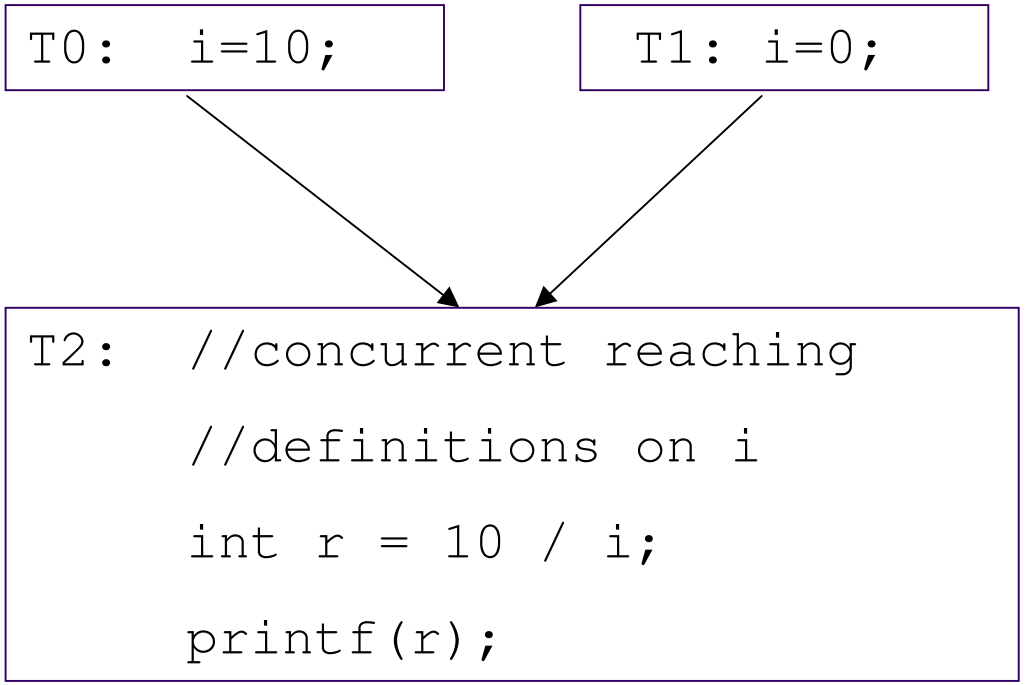


- Debugging and verification of a concurrent program is difficult due to the non-deterministic nature of concurrent executions.
- Common concurrent program bugs:
  - Deadlock
  - **Data Races**
    - Some data races might cause errors while some won't.
    - our research presents a *preliminary* result on a new approach for detecting data racings in a concurrent program. **(static analysis + dynamic verification)**

Data Race

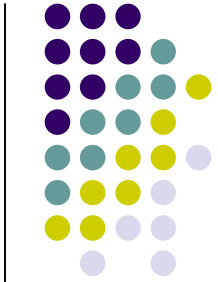


- Definition of shared variables may race against each other.



The value of i used for the division can be either 10 or 0.

Data Flow Analysis



- We want to discover concurrent reaching definitions in a program through data flow analysis.
  - CSSAME (Static Single Assignment for Mutual Exclusion) is a kind of data flow analysis output that contains the needed information.
  - **CSSAME is produced by Odyssey compiler (standard C -> CSSAME form).**

Static Single Assignment (SSA) Form

- Introduce a new variable for every definition (assignment).
  - every use is reached by exactly one definition.
- Introduce a phi function for every merge of control flows.

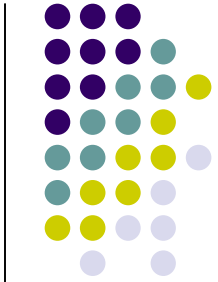
```
i = 10;
b = i;
i = i+3;
if (b > 10)
     b = b - 1;
else
     b = b + 1;
a = b + 1;
```

Original Code

```
i1 = 10;
b1 = i1;
i2 = i1+3;
if (b1 > 0)
     b2 = b1-1;
else
     b3= b1 + 1;
b4=phi(b2, b3);
a1 = b4 + 1;
```

SSA Form

Concurrent SSA (CSSA)



- SSA form for concurrent programs.
- Introduce a **pi** function to merge multiple data flows from multiple threads.

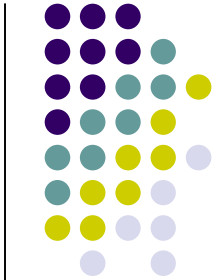
```
T0: i = 10;
    b = i + 2;

T1: i=9;
    printf("%d\n",i);
```

Original Code

```
T0: i1=10;
    i3=pi(i1, i2);
    b1=i3+2;

T1: i2=9;
    i4=pi(i1, i2);
    printf("%d\n",i4);
```

CSSA Code

CSSAME



- An extension of CSSA form to support mutual exclusion constructs.

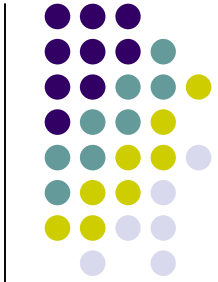
```
T0: lock(alock);
       i=10;
       b=i+2;
    unlock(alock);

T1: lock (alock);
       i=9;
    unlock (alock);
    printf("%d\n", i);
```

Original Code

```
T0: lock(alock);
       i1=10;

       b1=i1+2;
    unlock(alock);

T1:lock(alock);
       i2=9;
   unlock(alock);
   i3=pi(i1, i2);
   printf("%d\n",i3);
```

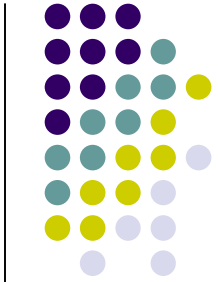
CSSAME Code

Usage of CSSAME



- Compiler statically analyzes the concurrent program.
  - Parameters of a **pi** function exposes the concurrent reaching definitions on the use of a variable.
  - Using **pi** functions, we can construct different value schedules for a concurrent program.

Construct Value Schedules from Pi function



```
T0: i1=10;
    i3=pi(i1, i2);
    b1=i3+2;

T1: i2=9;
    i4=pi(i1, i2);
    printf("%d\n",i4);
```

CSSA Code

value schedule-1: i3=i1, i4=i1;

value schedule-2: i3=i2, i4=i2;

value schedule-3: i3=i1; i4=i2;

value schedule-4: i3=i2; i4=i1;

A value schedule is the set (equivalence class) of all thread schedules that agree, for every critical read event r, on the critical write event that produced the value consumed by r. [Ur, 2003]

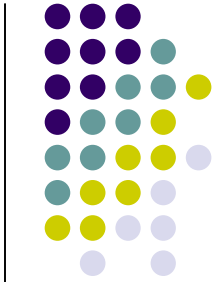
Value Schedule & Test Case

- Each value schedule corresponds to a way to **interleave** the assignments of a shared variable in concurrent execution.
- Running a concurrent program against a value schedule is equivalent to testing that program according to a set of execution interleaves.

Aspect Technology

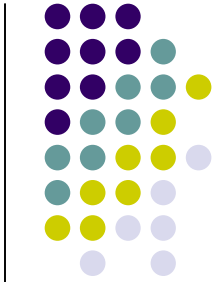
- Adding new functionalities into a program while preserving the integrity of original source code.
- Good for profiling and debugging.
- **We use it to manage the data flow of a program.**

A Side-Note



- For simplicity and clarity, we would like to
  - represent CSSAME form in Java
    - we assume there exists a compiler which compiles a piece of source code into CSSAME form in Java.
  - construct verification code in AspectJ

Aspect Technology

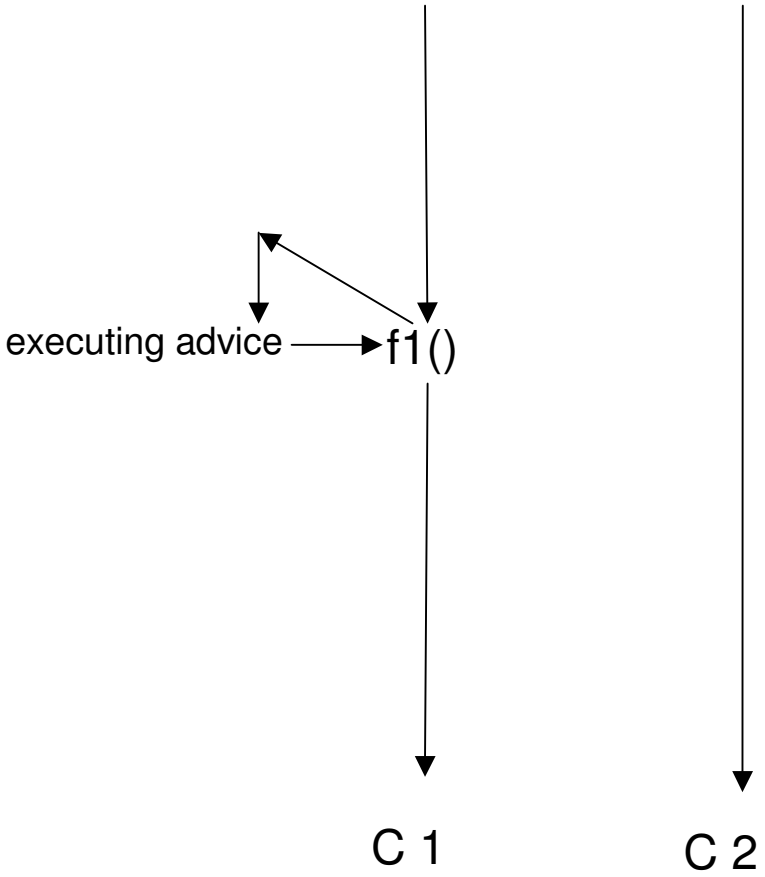


```
class Driver{
 public class C1 extends Thread{
     public void run(){f1();  }
     public void f1()  {;}
  }
 public class C2 extends Thread{
     public void run() {… }
  }
 public static void
   main(String[] args){
      Thread c1=new C1();
      Thread c2=new C2();
       c1.start(); c2.start();
} } }
```

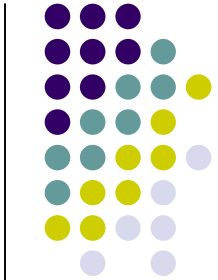
Implicitly invokes

```
aspect A{
pointcut delay_f1():
(call(public int
C1.f1()));

int before(): delay_f1(){
   sleep(3000);
   proceed();
  }
}
```

Execution Flow of Aspect

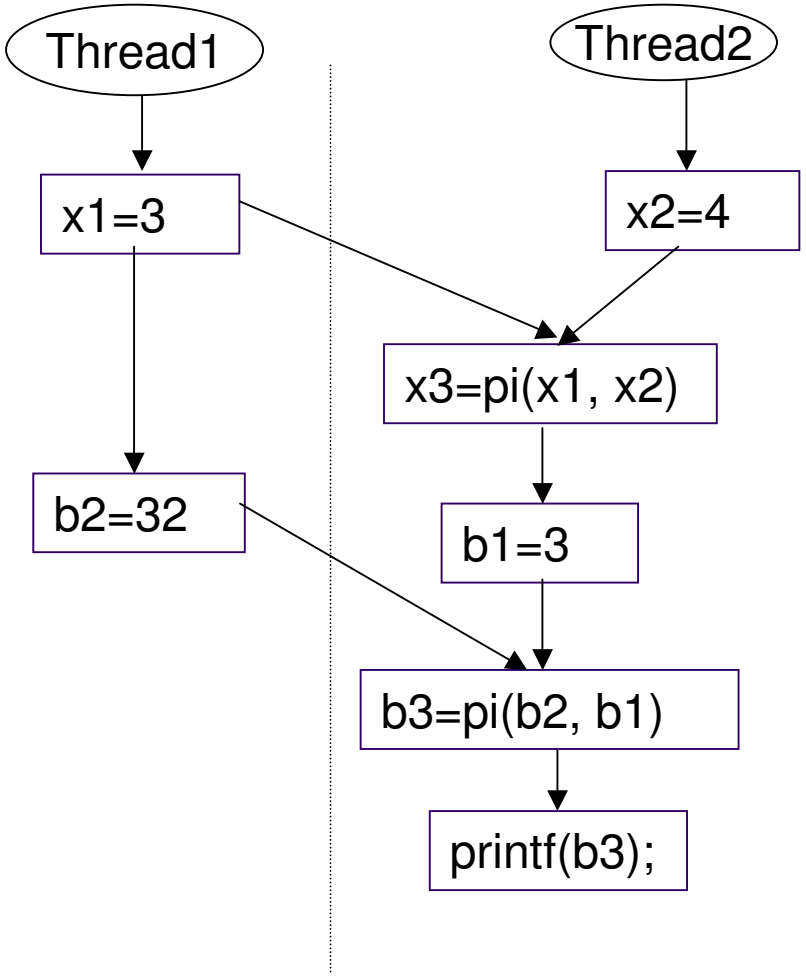


Choosing the Futures



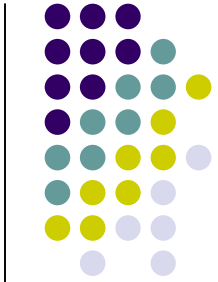
- Combining the strengths of both CSSAME and Aspect technology to enforce a concurrent execution to produce a data flow according to a selected value schedule.
  - futures stand for the potential values of a particular variable at a particular *use (in **pi** function)*
    - All possible futures (incoming definitions) are listed as the parameters of a **pi** function associated with that use.
  - Choosing the futures at a *use*
    - selecting an incoming definition from a set of definitions listed in the **pi** function's parameter list.

An Example



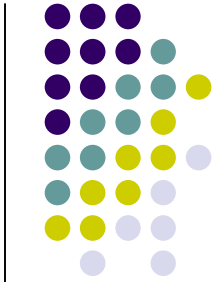
Value schedule x3=x1, b3=b2 is enforced by making pi(x1, x2) to return x2 and making pi(b1,b2) to return b2 using aspect advices.

Implementation



- Aspect pointcuts monitor two kinds of operation:
  - Invocation of **pi** functions
  - Writes to shared variables
- A particular value schedule is enforced by choosing desirable values from **pi** functions.
  - When a **pi** function is encountered during the execution, the aspect advice of that **pi** function is executed to return the value of one of definitions stored in the parameter list of the pi function, according to the value schedule.
  - The execution of an advice of a **pi** function will be stalled until the necessary value is available.

Aspect in Action



```
class Driver{
   int x1, x2;
   public void exec(){
      Thread t1=new Thread();
      Thread t2=new Thread();
      t1.start();   t2.start();
   }
   class T1 extends Thread{
   …x1=1…. }
   class T2 extends Thread{
        public void run(){
        …..
        x2=4;
        x3=pi_x3(x1, x2);
        ….
      }
    }
    int pi_x3(int a, int b){
      return -1;//dummy return
    }
}
```

Original Code

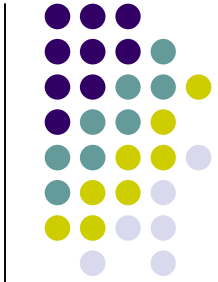
implicitly invokes

```
pointcut on Driver.x1;

advice on Driver.x1{

//set the status of //Driver.x1
to ready

//store the assigned value //of
Driver.x1

//notifyall

}
```

implicitly invokes

```
pointcut for Driver.pi_x3(int a,
int b);

advice for Driver.pi_x3(int a,
int b){

//check for the readiness of
//selected variable

//if it is set → return its
//value

//if it is not yet set→ wait

}
```

Implementing Pi function

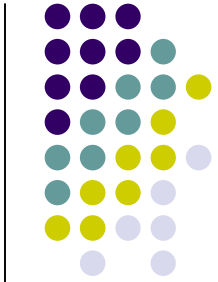


- Code for selecting x1 from pi function: x3=pi(x1, x2)

```
pointcut pi_x3_call(int x1, int x2):
(call(private int Thread1.pi_x3(int,
int)) && args(x1, x2));

int around(int x1, int x2): args(x1, x2)
&& pi_x3_call(int, int){
    ItemWrapper choice=obj_x1;
    synchronized(choice) {
    //if selected object is not
    //wait
       while (!choice.ready)
            choice.wait();
        }
    return choice.value;
 }
```

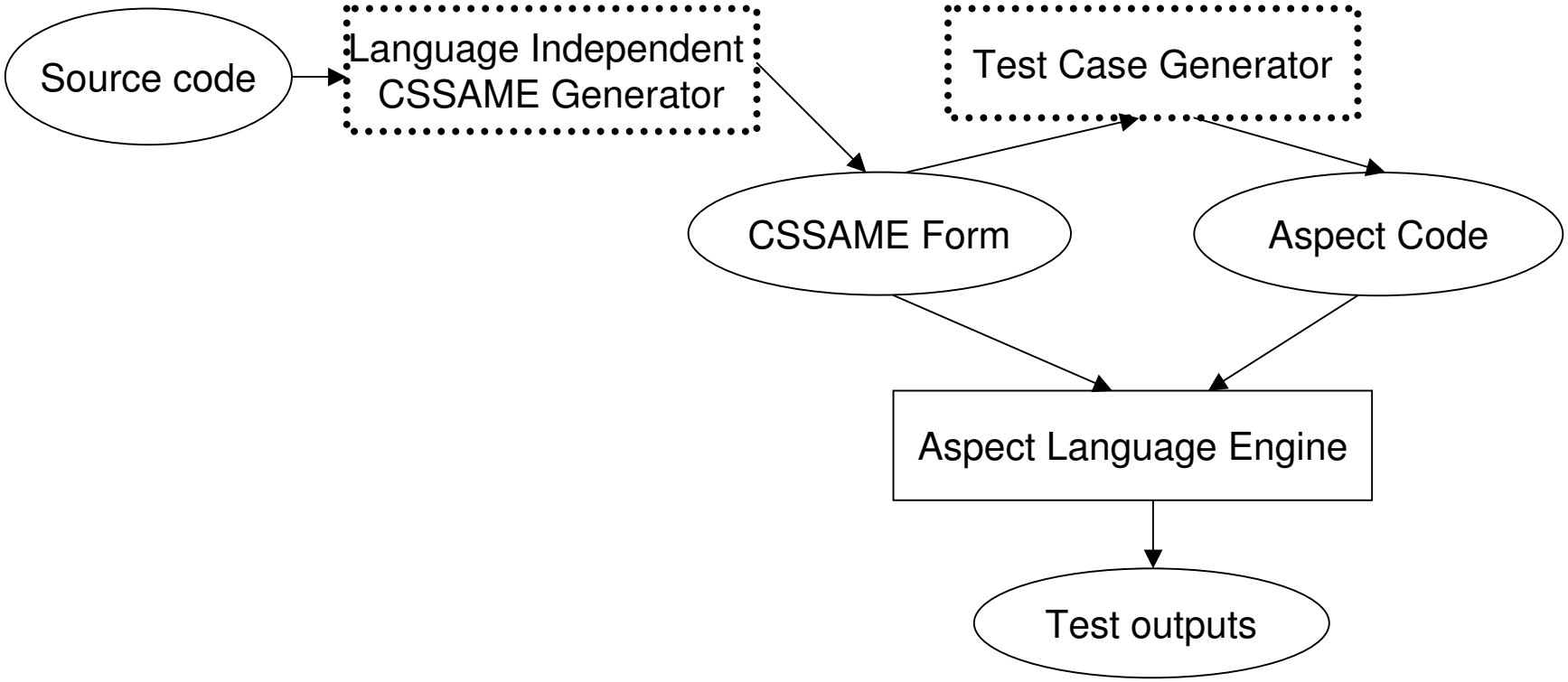
```
pointcut set_x1(int n):
set(private int
Driver.x1) && args(n);

after(int n): set_x1(n){
  synchronized(obj_x1){
   obj_x1.ready=true;
   obj_x1.value=n;
   obj_x1.notifyAll();
  }
}
```

Verification for Correctness

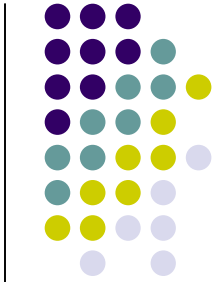


- The correctness of a concurrent program is verified by running it against value schedules.
- A value schedule that leads to an error can be further examined and incrementally debugged.

Sketch for a Complete Verification Process

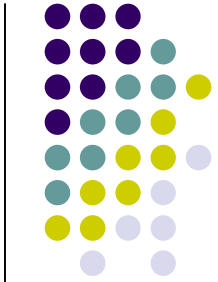


Limitations



- Odyssey compiler is not language independent. (C only)
- May produce inconsistent value schedules.
- Odyssey compiler is not capable of performing inter-procedure analysis.

Inconsistent Value Schedule



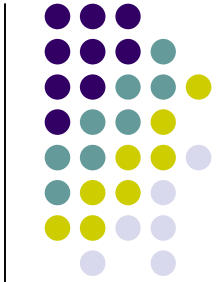
- picking an “inconsistent” value schedules which produces **deadlock**
  - picking y2=y1 at Thread 2 while x3 actually resolved to x1 in Thread 1produces an inconsistent value schedule.

```
T0: y0=33;
……
```

```
T1:x1=1;
    if (x1 > 3){
      x2=5;
      y1=4;
    x3 = phi(x1,x2);
```

```
T2:y2=pi(y0,y1);
……
```

Inconsistent Value Schedule

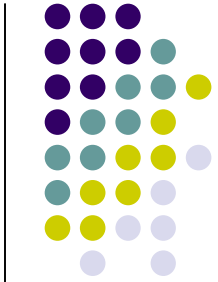


- picking an “inconsistent” value schedule which produces **no deadlock**
  - Picking a3=a1and b3=b2 at Thread 2 produces an inconsistent value schedule.
  - However, the selected schedule produces values and output which cannot happen in any valid schedule.

```
T0: lock(L)
     a1 = 1
     b1 = 1
    unlock(L)
```

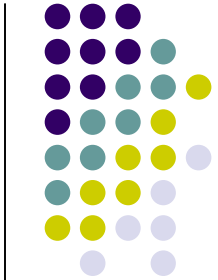
```
T1:int z0 = 0
   lock(L)
     a2 = 2
     b2 = 2
   unlock(L)
   lock(L)
     a3 = pi(a1, a2)
     if (a3 == 1)
       b3 = pi(b1, b2)
       z1 = b3
     z2 = phi(z0, z1)
     print(a3, z2)
   unlock(L)
```

Future Works



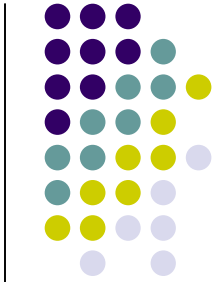
- Modify an existing language-neutral compiler, such as gcc, to produce the CSSAME form.
- Implementing inter-procedural analysis.
- Managing the test cases
  - Automatically generates and runs test cases
  - Does not generate test cases derived from inconsistent value schedules
- Building tool support.
- Collecting benchmarks.

Conclusion



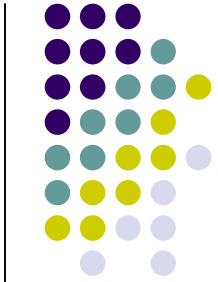
- This is a report on a preliminary research.
- Our approach provides a basis to construct a deterministic debugging environment for a concurrent program by combining:
  - Results from static data flow analysis.
  - Deterministic execution of a value schedule controlled by the aspect.

Finally



- Thank you.
  - And
- Questions?

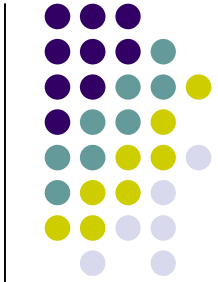
Implementing Phi Function



```
pointcut phi_b3_call(int b1, int b2):(call(private int
Thread2.phi_b3(int, int)) && args(b1, b2));

int around (int a, int b): args(a, b) && phi_b3_call(int, int){
   int i=obj_b.value;
   return i;
}
```

```
pointcut set_b1(int n): set(private int
Thread1.d_b1) && args(n);
after (int n): set_b1(n){
    obj_b.value=n;
}
pointcut set_b2(int n): set(private int
Thread1.d_b2) && args(n);
after (int n): set_b2(n){
    obj_b.value=n;
}
```

Usage of CSSAME



- Moreover, CSSAME serves as an intermediate form which can be used for debugging.
  - The execution sequence of a program in CSSAME form can be controlled to follow a particular value schedule using Aspect technology.