Differential Symbolic Execution

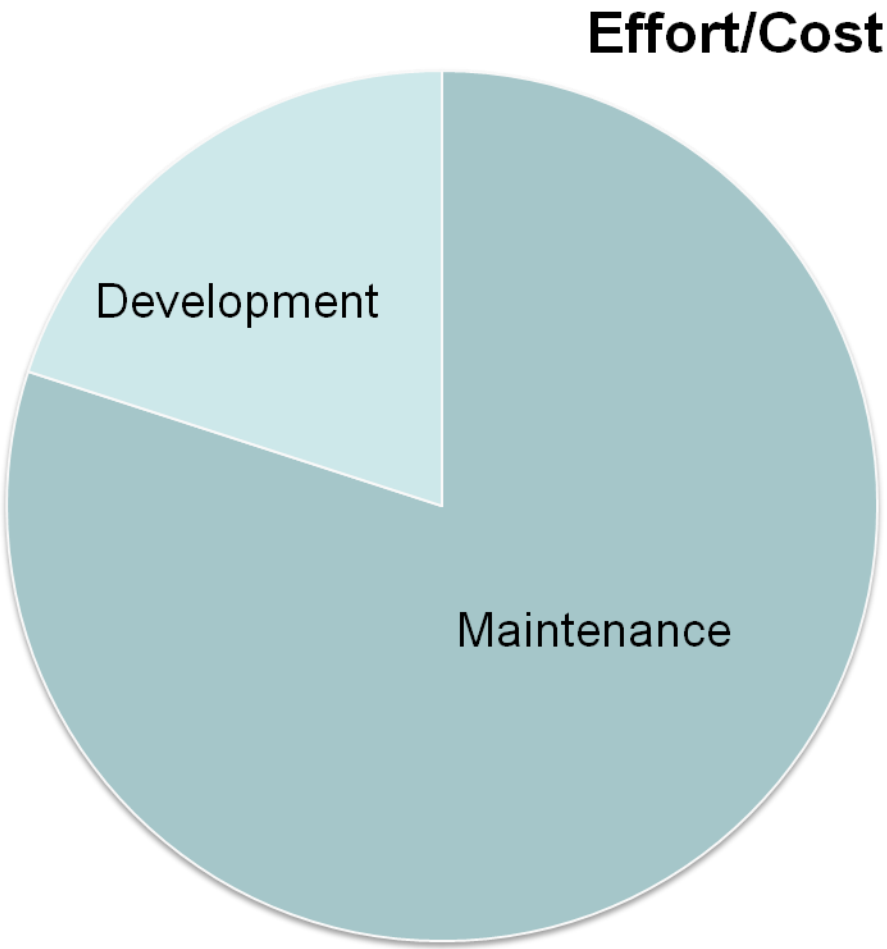
Suzette Person, Matthew B. Dwyer, Sebastian Elbaum
University of Nebraska – Lincoln

Corina Păsăreanu
NASA Ames Research Center

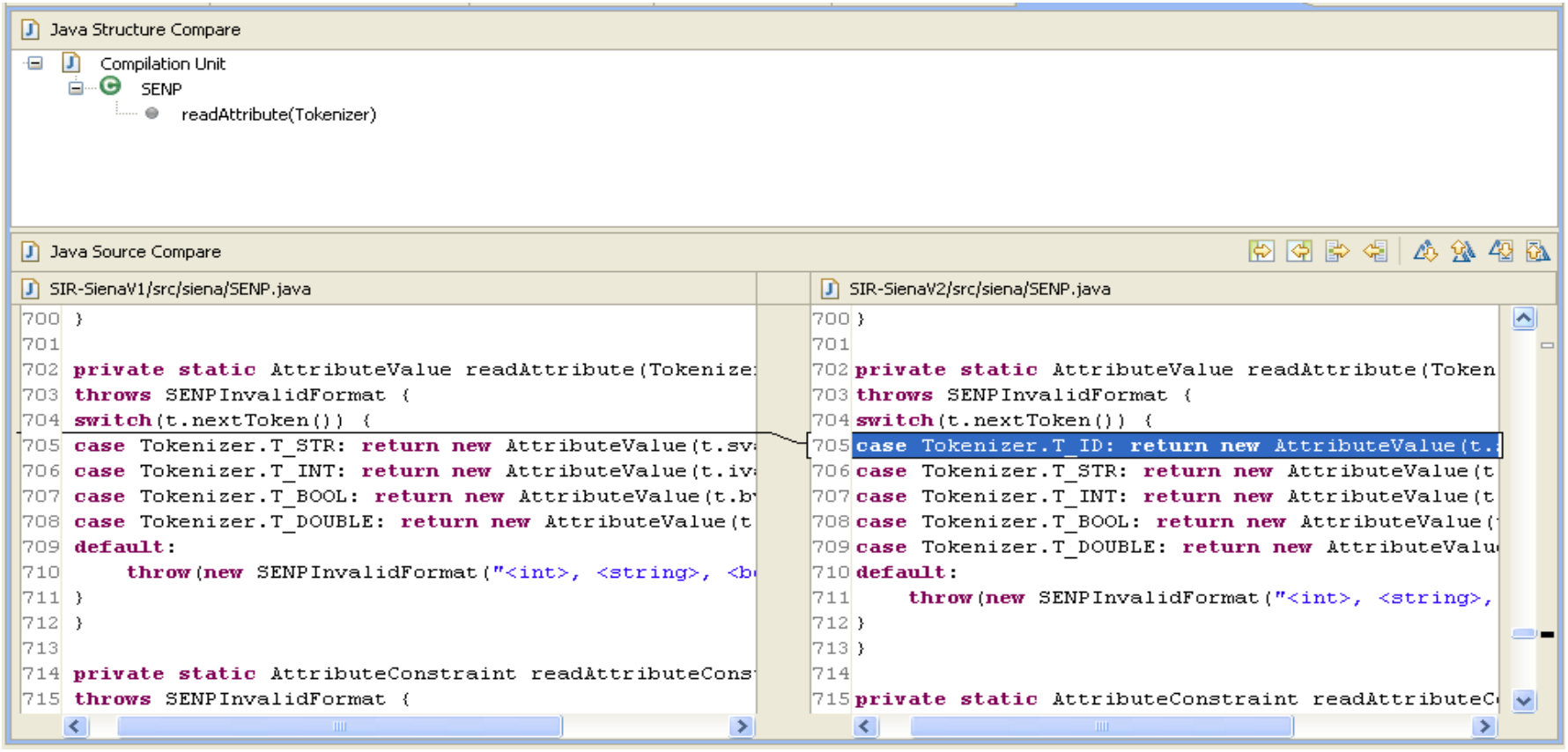
Funded in part by the National Science Foundation and NASA EPSCoR

Motivation

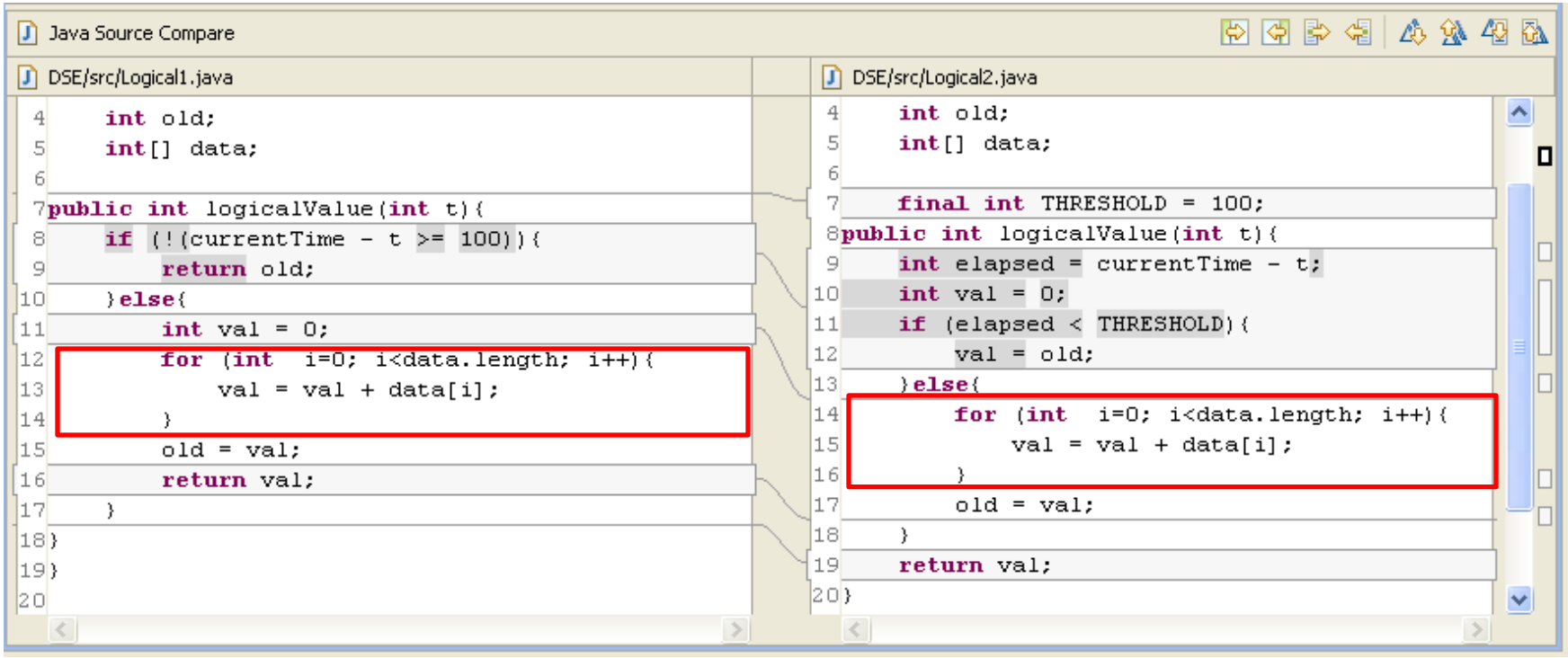
- Locate and fix faults
- Refactor code
- Extend functionality
- Merge versions
- …



Motivation


Java Structure Compare
Compilation Unit
SENP
readAttribute(Tokenizer)
Java Source Compare
SIR-SienaV1/src/siena/SENP.java
SIR-SienaV2/src/siena/SENP.java
700 }
701
702 private static AttributeValue readAttribute(Tokenize
703 throws SENPInvalidFormat {
704 switch(t.nextToken()) {
705 case Tokenizer.T_STR: return new AttributeValue(t.sv
706 case Tokenizer.T_INT: return new AttributeValue(t.iv
707 case Tokenizer.T_BOOL: return new AttributeValue(t.b
708 case Tokenizer.T_DOUBLE: return new AttributeValue(t
709 default:
710 throw(new SENPInvalidFormat("<int>, <string>, <b
711 }
712 }
713
714 private static AttributeConstraint readAttributeCons
715 throws SENPInvalidFormat {
700 }
701
702 private static AttributeValue readAttribute(Token
703 throws SENPInvalidFormat {
704 switch(t.nextToken()) {
705 case Tokenizer.T_ID: return new AttributeValue(t.
706 case Tokenizer.T_STR: return new AttributeValue(t
707 case Tokenizer.T_INT: return new AttributeValue(t
708 case Tokenizer.T_BOOL: return new AttributeValue(
709 case Tokenizer.T_DOUBLE: return new AttributeValu
710 default:
711 throw(new SENPInvalidFormat("<int>, <string>,
712 }
713 }
714
715 private static AttributeConstraint readAttributeC

Motivation




```
4     int old;
5     int[] data;
6
7  public int logicalValue(int t){
8     if (!(currentTime - t >= 100)){
9         return old;
10    }else{
11        int val = 0;
12        for (int  i=0; i<data.length; i++){
13            val = val + data[i];
14        }
15        old = val;
16        return val;
17    }
18 }
19 }
20
```

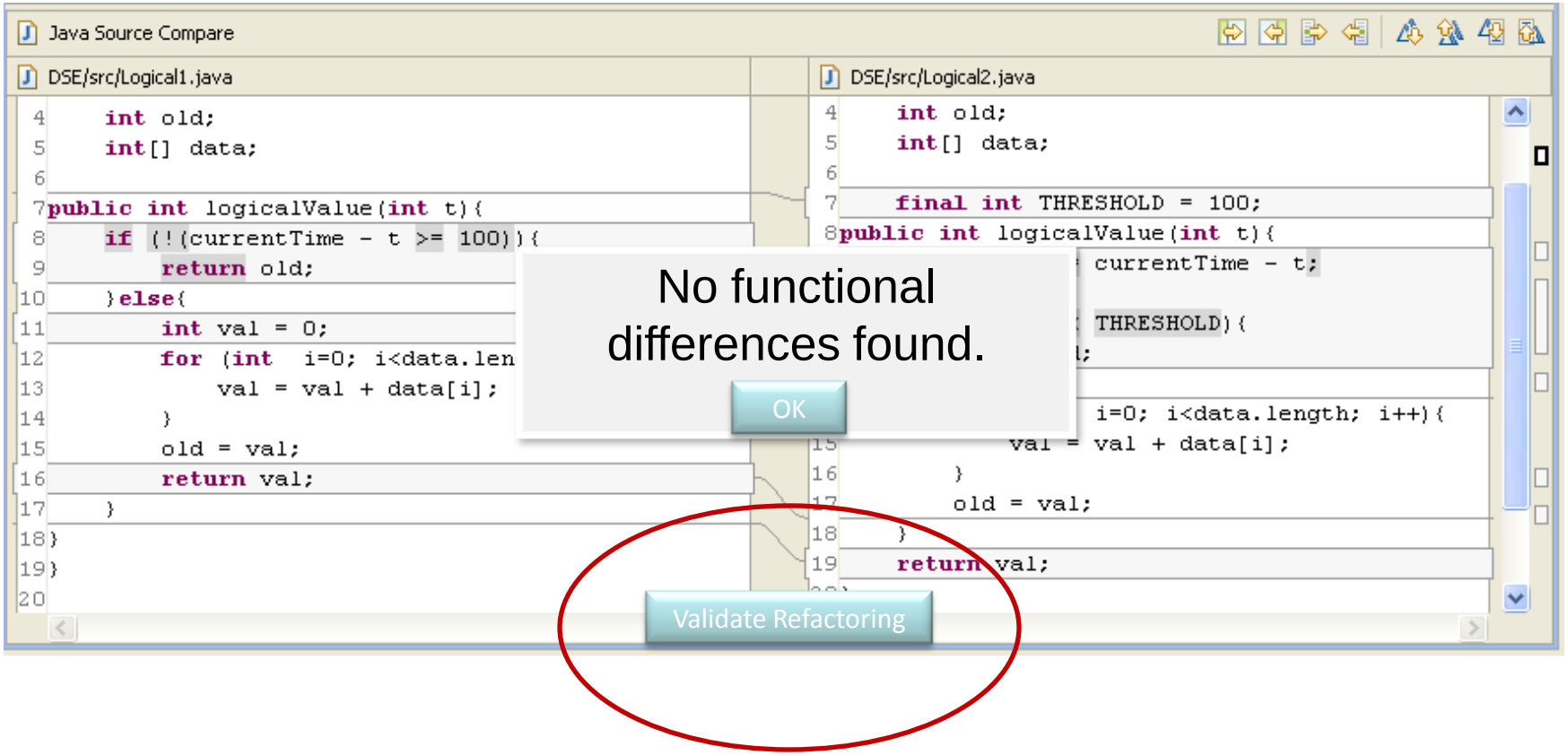
```
4     int old;
5     int[] data;
6
7     final int THRESHOLD = 100;
8  public int logicalValue(int t){
9     int elapsed = currentTime - t;
10    int val = 0;
11    if (elapsed < THRESHOLD){
12        val = old;
13    }else{
14        for (int  i=0; i<data.length; i++){
15            val = val + data[i];
16        }
17        old = val;
18    }
19    return val;
20 }
```

Differential Symbolic Execution (DSE)

- Detect and characterize the effects of program changes in terms of *behavioral* differences between program versions

Symbolic Execution + Over-approximating Symbolic Summaries

Differential Symbolic Execution (DSE)



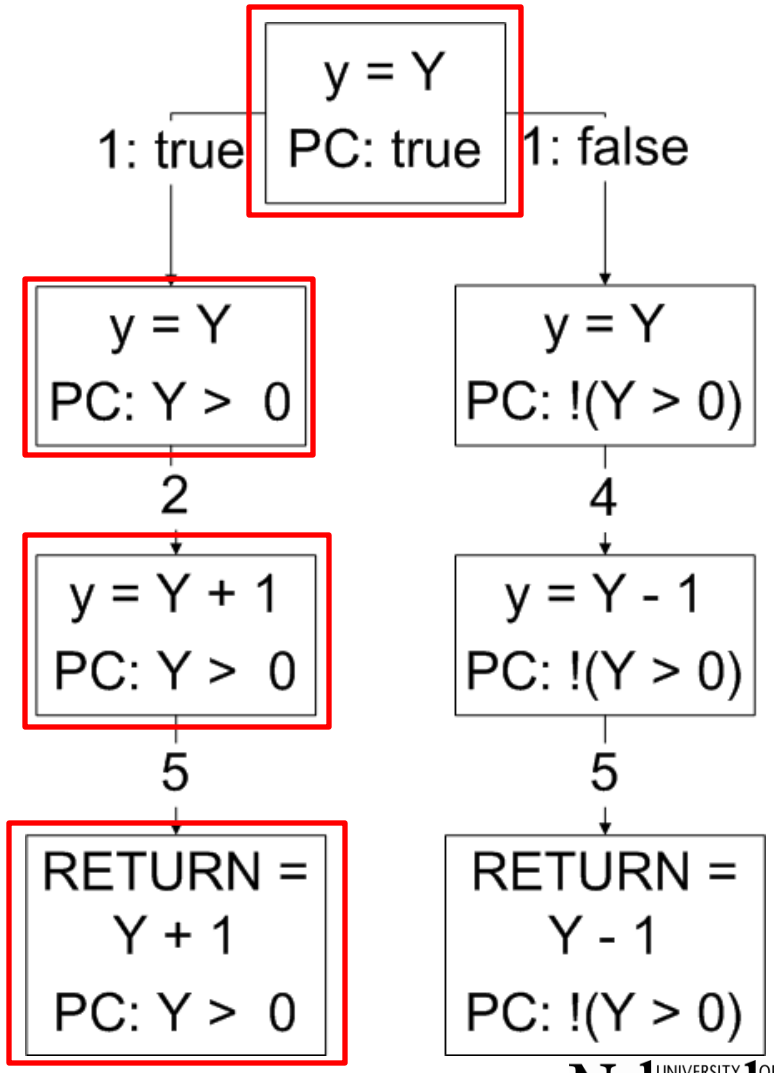
Overview of Presentation

- DSE methodology
- Summaries of program behavior
- Notions of equivalence and deltas
- Applications of DSE
- Related work
- Conclusions and future work

Symbolic Execution

```
int m(int y){
1:  if (y > 0)
2:      y++;
3:  else
4:      y--;
5:  return y;
}
```

m_{sum} =
{(Y > 0, RETURN==Y+1)}



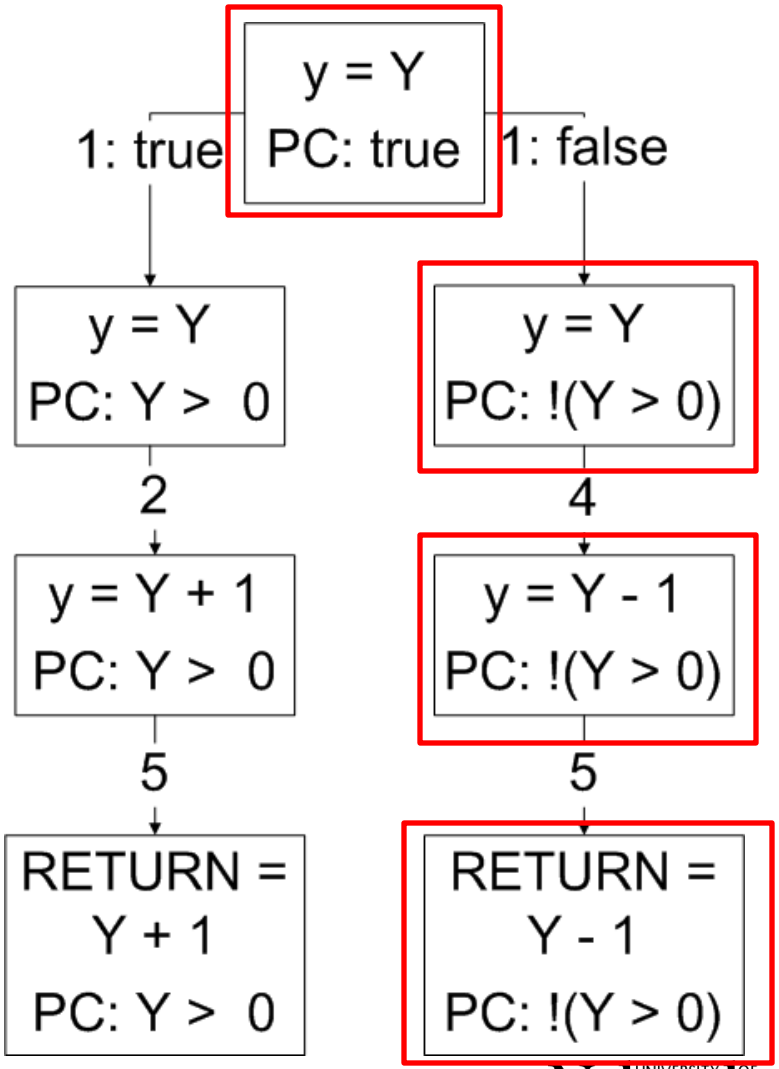
Symbolic Execution

```
int m(int y){
1:  if (y > 0)
2:      y++;
3:  else
4:      y--;
5:  return y;
}
```

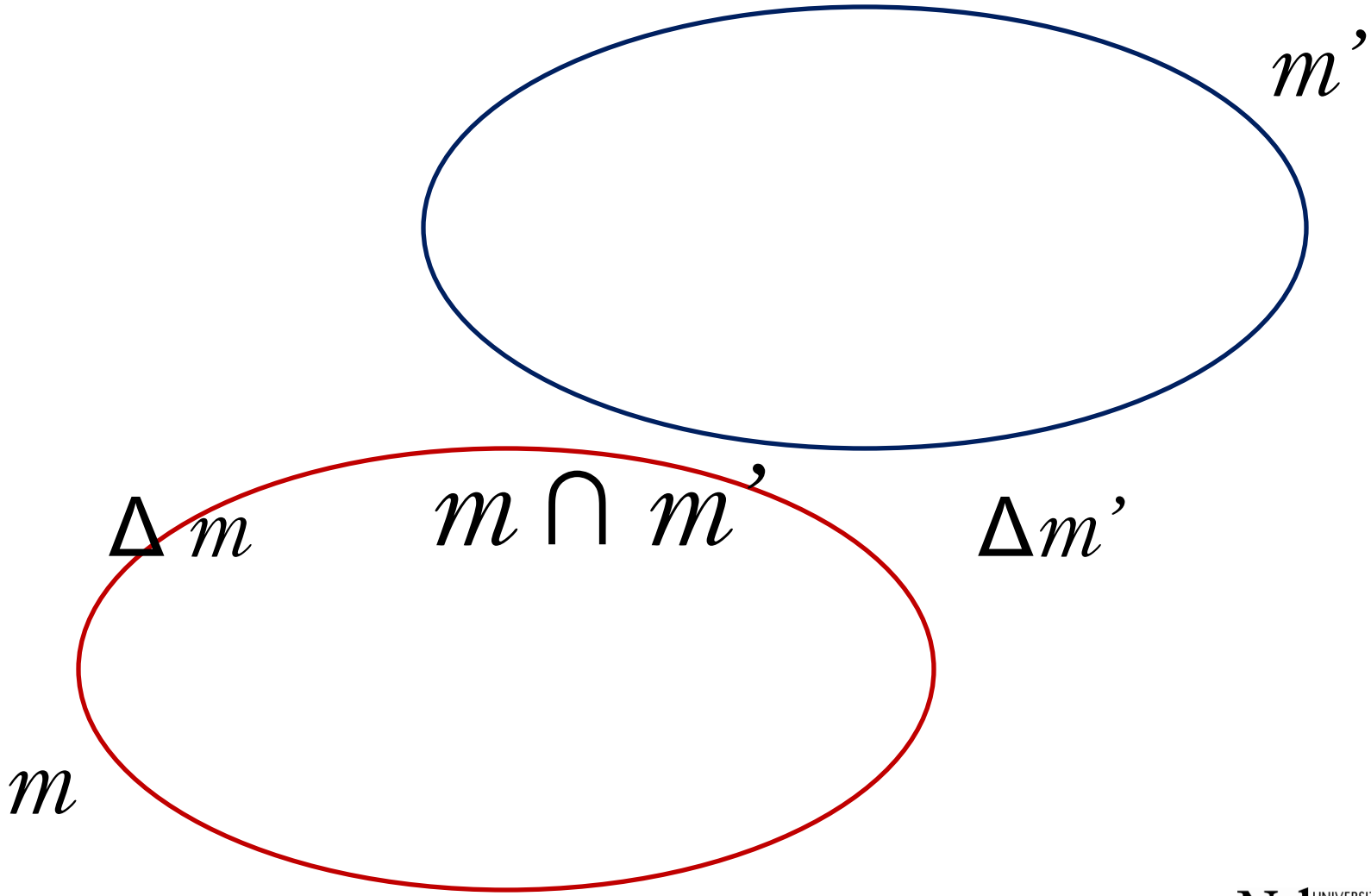
m_{sum} =

{(Y > 0, RETURN==Y+1)}

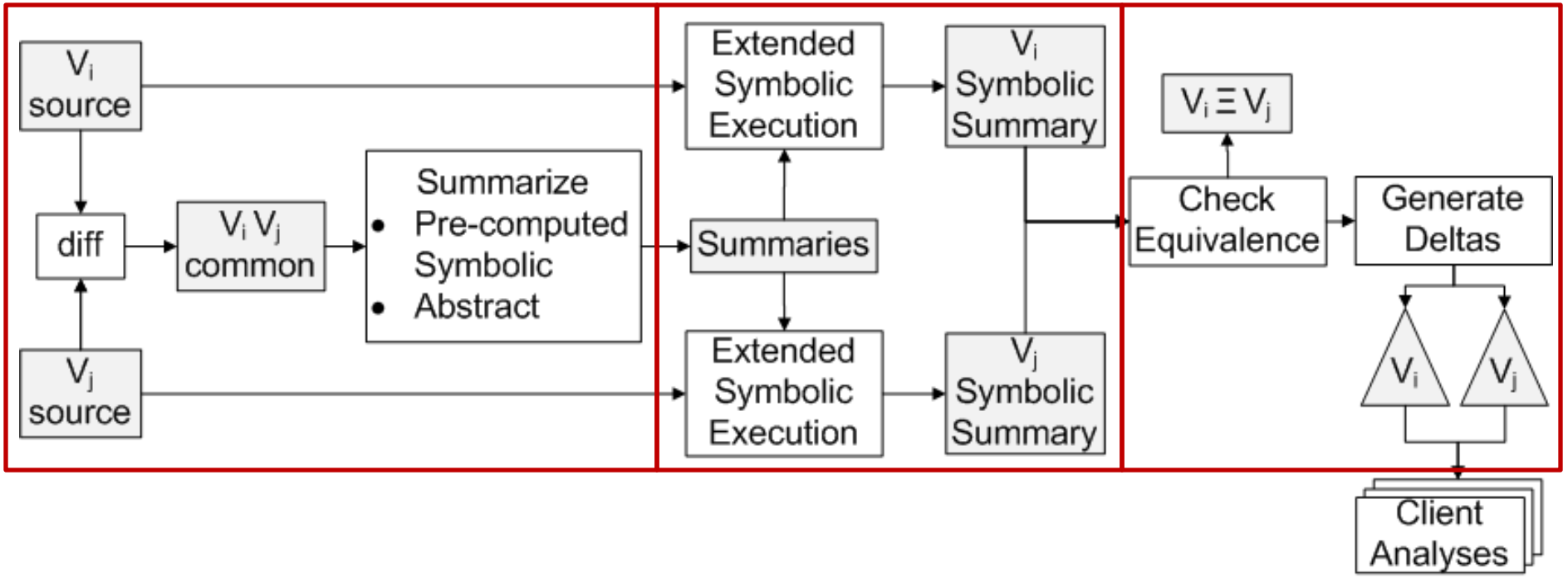
{!(Y > 0), RETURN==Y-1)}



Differential Symbolic Execution

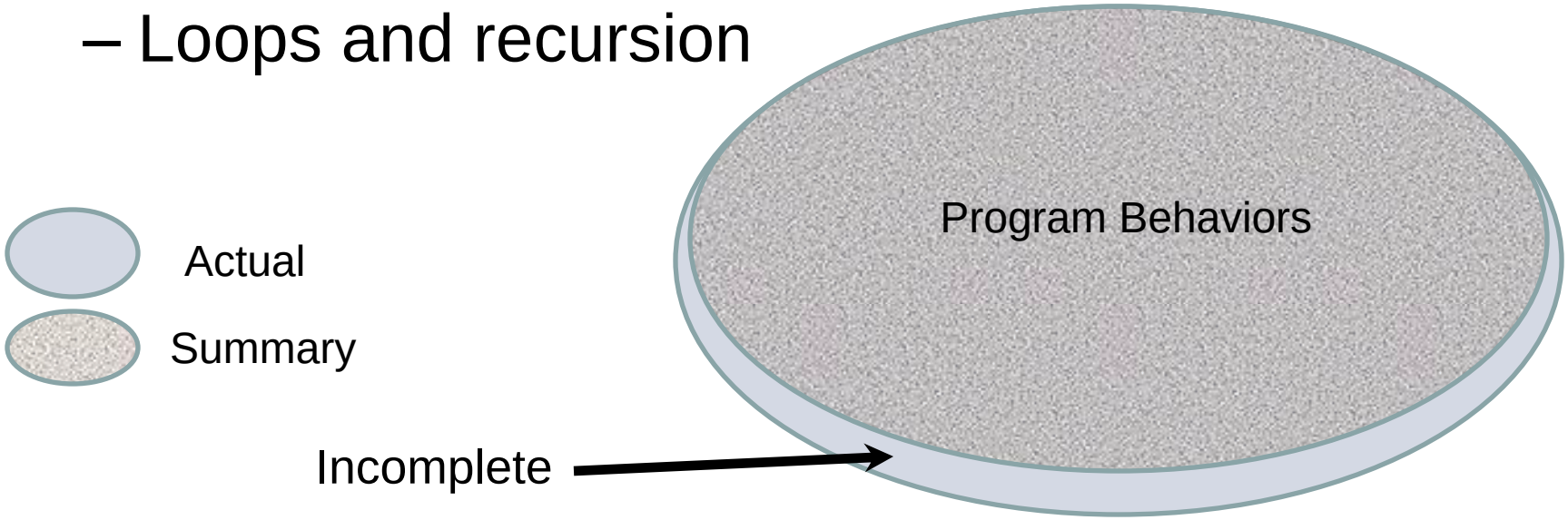


Differential Symbolic Execution

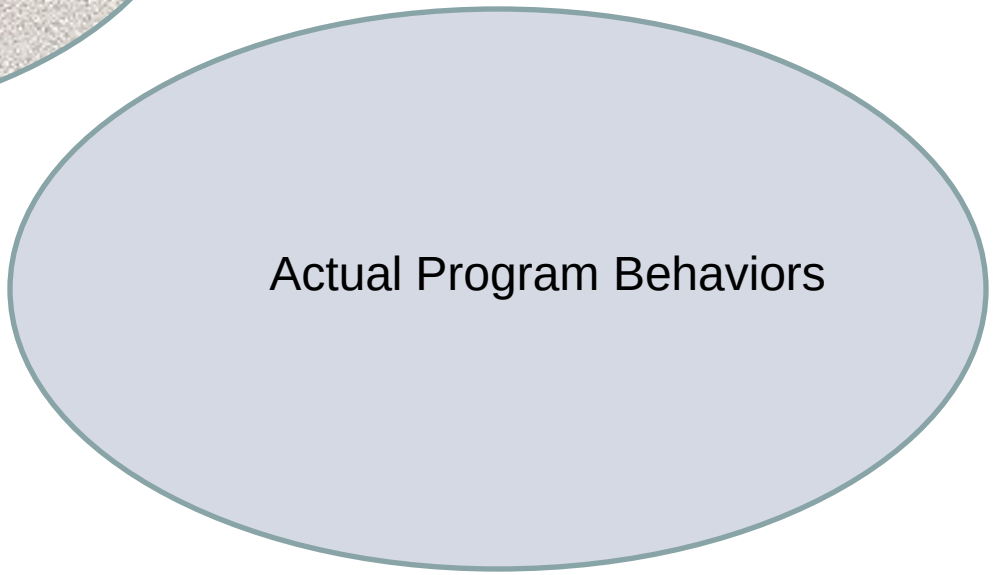


Incomplete Summaries

- It is not always possible to compute complete summaries
  - Non-linear arithmetic
  - Loops and recursion



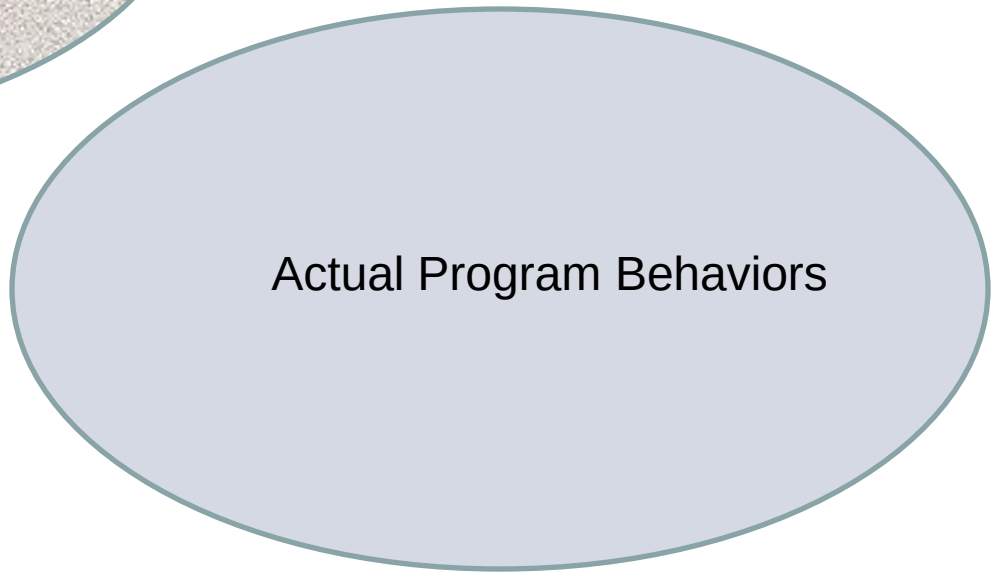
Incomplete Summaries



Is all of the input space accounted for by the summary?

Incomplete Summaries

$m_{sum} = \{(i_1, e_1), (i_2, e_2), (i_3, e_3)\}$

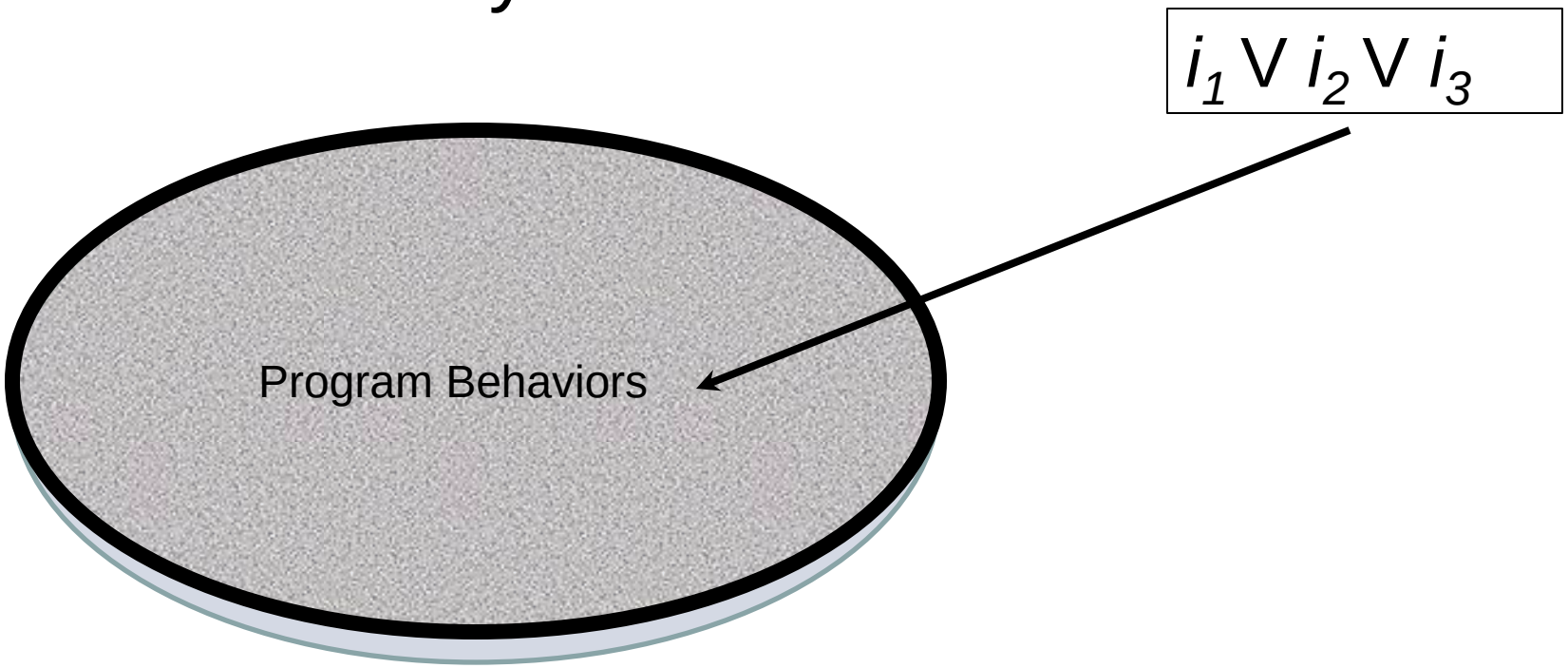


Is the disjunction of inputs valid?

$$i_1 \vee i_2 \vee i_3$$

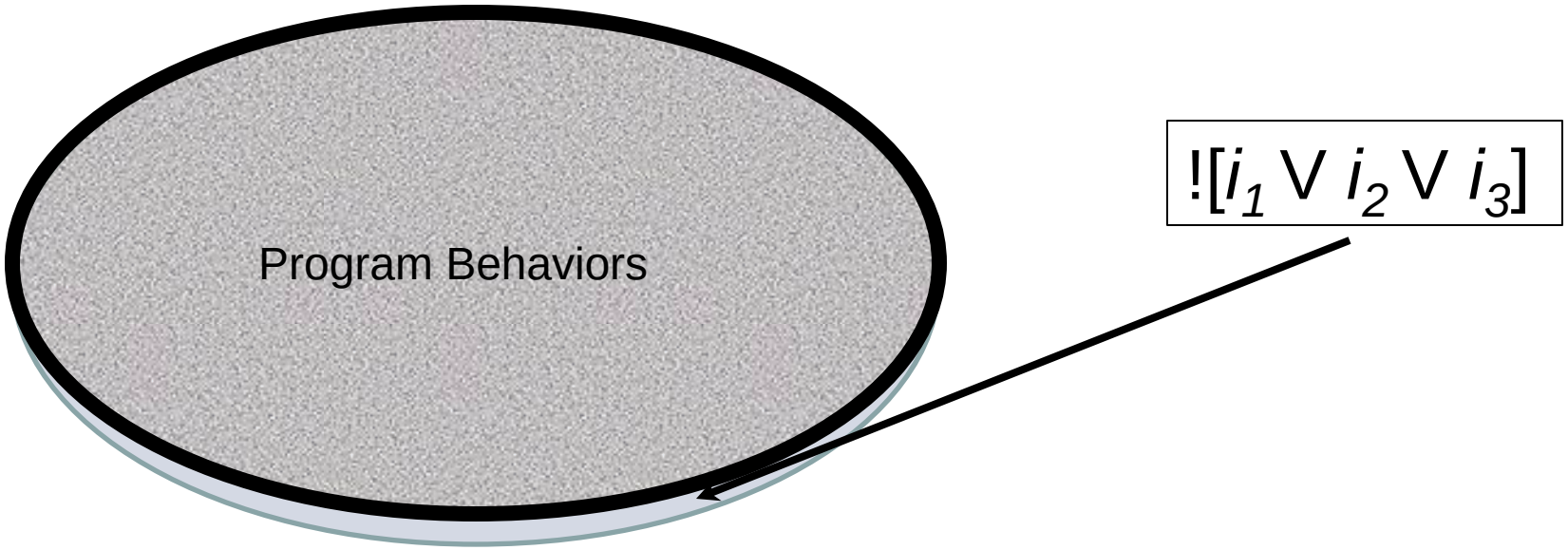
Incomplete Summaries

Explicitly define the input space covered by the summary



Incomplete Summaries

Focus subsequent analysis tool on behaviors not covered



Abstract Summaries on Common Blocks

```
void test(){  //v1
  s1;
  s2;
  …
  for (int i=0; i<len; i++;){
     val = val + x[i];
  }
  old = val;
  sn;
  …
}
```

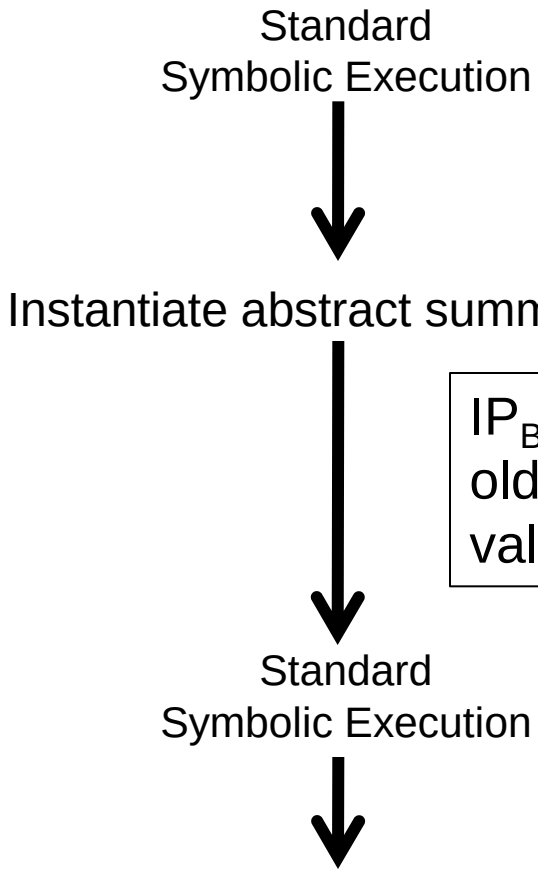
Abstract Summary
Read set:{x,val}
Write set:{val,old}

```
void test(){  //v2
  sa;
  sb;
  …
  for (int i=0; i<len; i++;){
     val = val + x[i];
  }
  old = val;
  sm;
  …
}
```

Boolean IP_B(int[] X, int val)
int old_B(int[] X, int val)
int val_B(int[] X, int val)

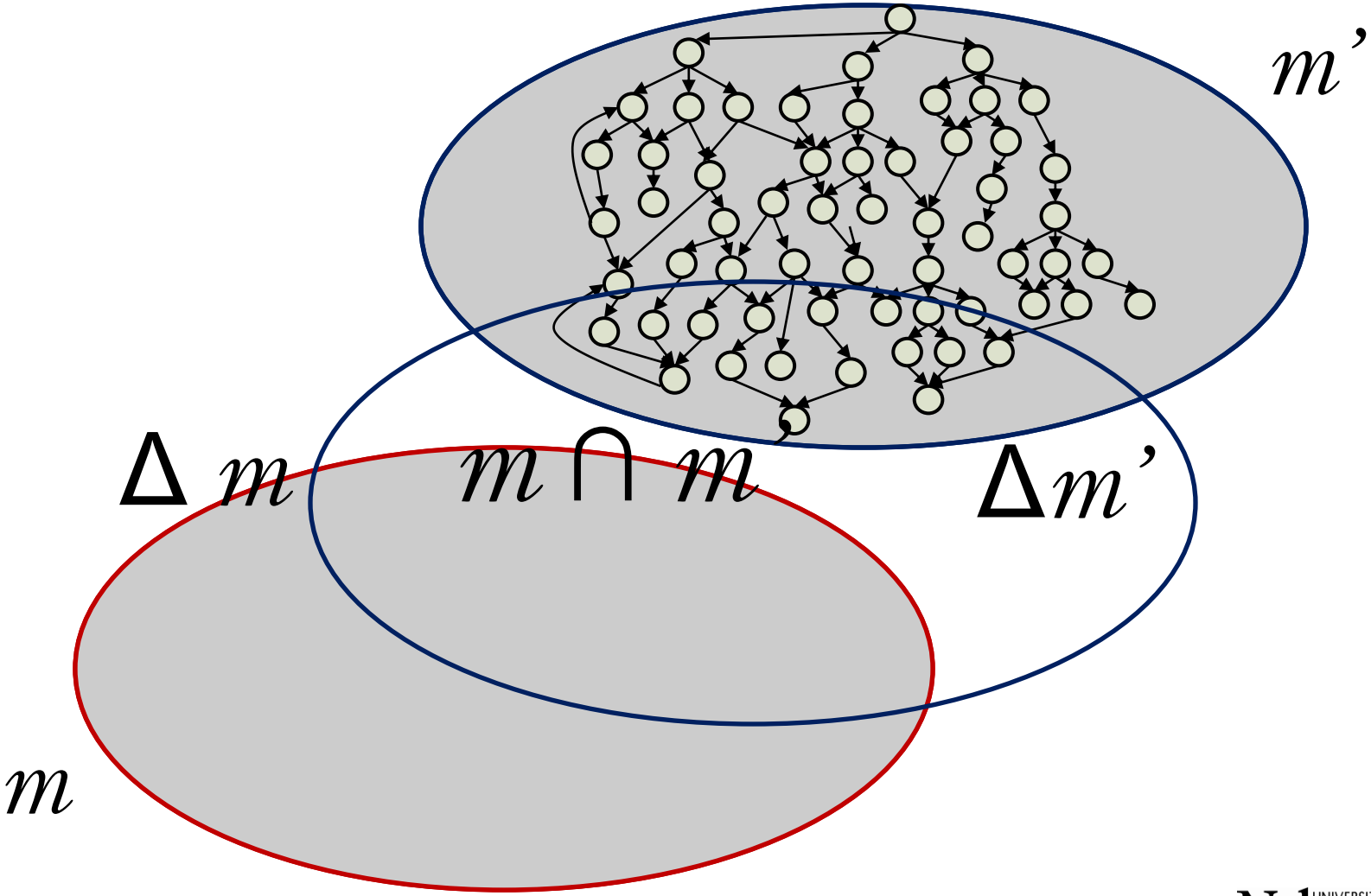
Abstract Summaries on Common Blocks

```
void test(){  //v1
  s1;
  s2;
  …
  for (int i=0; i<len; i++;){
     val = val + x[i];
  }
  old = val;
  sn;
  …
}
```

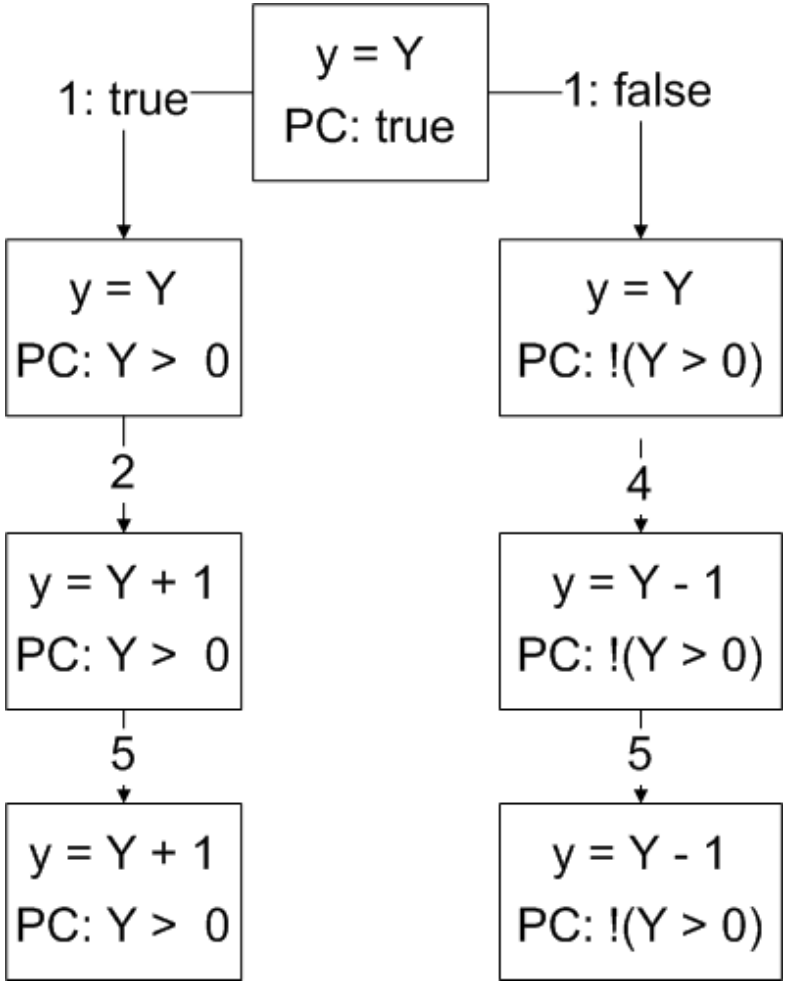


$IP_B(x,val)$
$old == old_B(x,val)$
$val == val_B(x,val)$

Functional Equivalence



Functional Equivalence



m_{sum}=
{(Y > 0, RETURN==Y+1),
(!(Y > 0), RETURN==Y-1)}

↓

m_{sum}=
(Y > 0 ∧ RETURN==Y+1)
∨
(!(Y > 0) ∧ RETURN==Y-1)

Functional Equivalence

```
int m(int y){//v1
1:  if (y > 0)
2:      y++;
3:  else
4:      y--;
5:  return y;
  }
```

```
int m(int y){//v2
1:  if (y <= 0)
2:      y--;
3:  else
4:      y++;
5:  return y;
  }
```

((Y > 0 ∧ RETURN==Y+1) ∨ (!(Y > 0) ∧ RETURN==Y-1))

?
≡

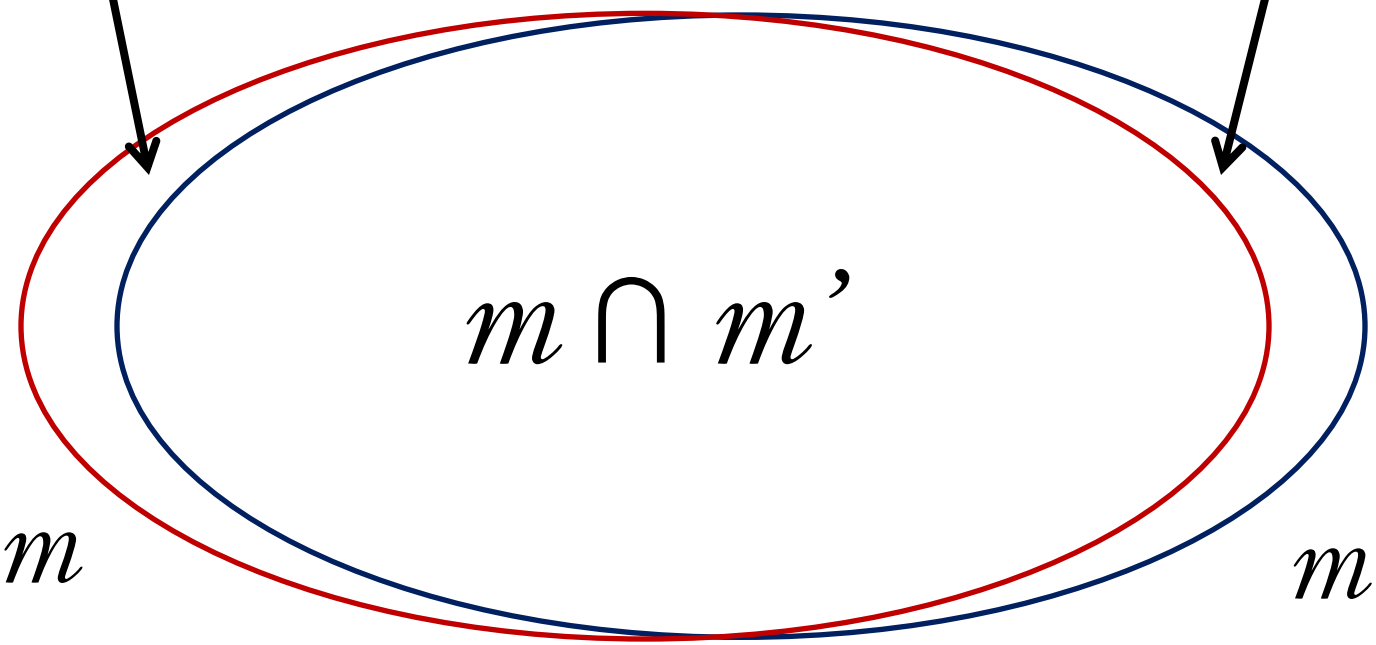
((Y <= 0 ∧ RETURN==Y-1) ∨ (!(Y <= 0) ∧ RETURN==Y+1))

Functionally Equivalent? ✓

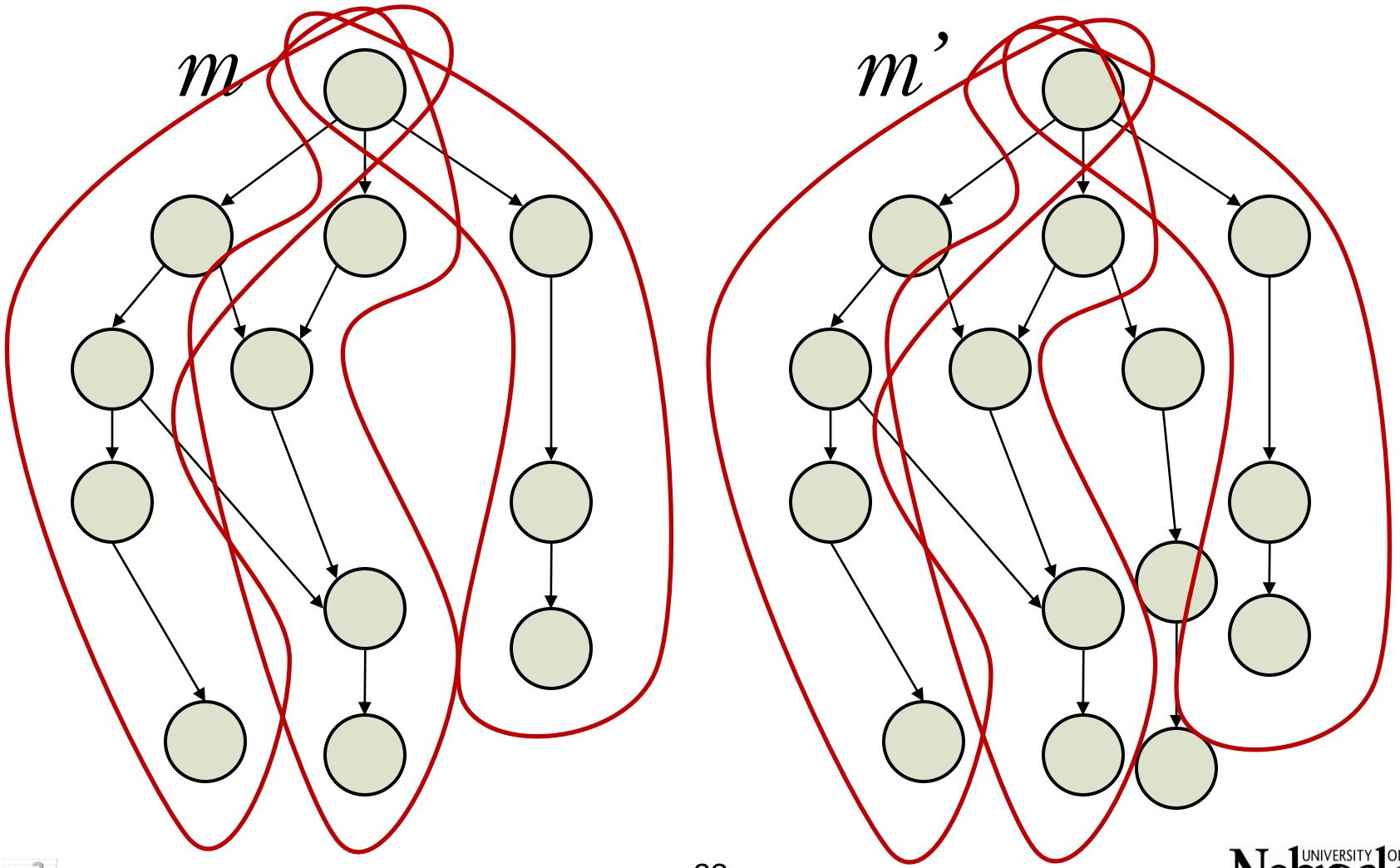
Functional Deltas

$\Delta m = m \wedge \neg m'$

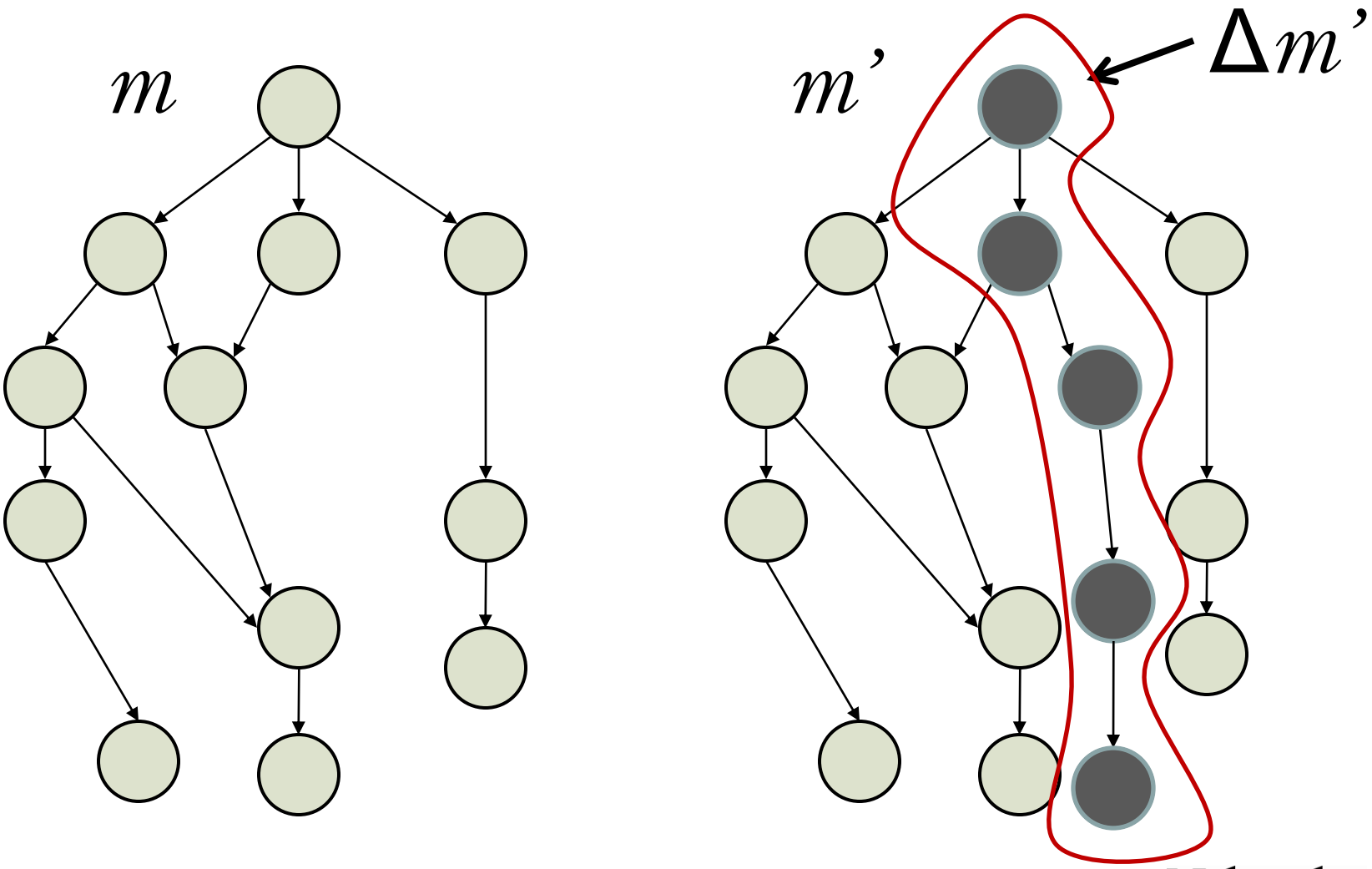
$\Delta m' = m' \wedge \neg m$



Partition-effects Equivalence



Partition-effects Delta



Application of DSE

- Prototype based on Symbolic PathFinder (JPF) & CVC3 theorem prover
- Applied to artifacts from SIR
  - JMeter
  - Siena
- Client applications
  - Refactoring assurance
  - Test suite evolution
  - Change characterization

Change Characterization

```
//Siena version 3
public static boolean match(byte[] x, byte[] y){
  if (x.len != y.len) return false;
  for(int i=0; i<x.len; ++i)
     if (x[i] != y[i]) return false;
  return true;
}
```

```
//Siena version 4
public static boolean match(byte[] x, byte[] y){
  if (x == null && y == null) return true;
  if (x == null || y ==null || x.len != y.len) return false;
  for(int i=0; i<x.len; ++i)
     if (x[i] != y[i]) return false;
  return true;
}
```

Change Characterization

match() Version 3	
Input Partition	Effect
X==null	RETURN == EXCEPTION
Y==null	RETURN == EXCEPTION
!(X==null) ∧ !(Y==null) ∧ (X.I !=Y.I)	RETURN == FALSE
!(X==null) ∧ !(Y==null) ∧ (X.I !=Y.I) ∧ $IP_{B1}(T, X, Y)$	RETURN == $RET_{B1}(T, X, Y)$

match() Version 4	
X==null ∧ Y==null	RETURN == TRUE
X==null ∧ !(Y==null)	RETURN == FALSE
!(X==null) ∧ Y==null	RETURN == FALSE
!(X==null) ∧ !(Y==null) ∧ (X.I !=Y.I)	RETURN == FALSE
!(X==null) ∧ !(Y==null) ∧ (X.I !=Y.I) ∧ $IP_{B1}(T, X, Y)$	RETURN == $RET_{B1}(T, X, Y)$

Change Characterization

On input	match() Version 3	match() Version 4
x == null Λ y == null	throws NRE	RETURN == TRUE
x == null Λ y !=null	throws NRE	RETURN == FALSE
x != null Λ y == null	throws NRE	RETURN == FALSE

Related Work

- Jackson et al. (ICSM ‘94)
- Neamtiu et al. (MSR ‘05)
- Apiwattanapong et al. (ASE ‘07)
- Santelices et al., Apiwattanapong et al. (ASE ‘08, TAIC PART ‘06)
- Siegel et al. (ISSTA ’06, PVM/MPI ‘08)
- Notkin (PASTE ‘02)

Conclusion

- Differential symbolic execution precisely detects and characterizes behavioral differences between two program versions
  - Functional equivalence and deltas
  - Partition-effects equivalence and deltas
- DSE leverages program commonalities to address summary completeness

Future Work

- Further explore DSE algorithms and extend theoretical foundations
- Automate support for client applications
- Study the cost and effectiveness of DSE in automating software maintenance tasks

Differential Symbolic Execution

Suzette Person, Matthew B. Dwyer, Sebastian Elbaum
University of Nebraska – Lincoln

Corina Păsăreanu
NASA Ames Research Center

Funded in part by the National Science Foundation and NASA EPSCoR