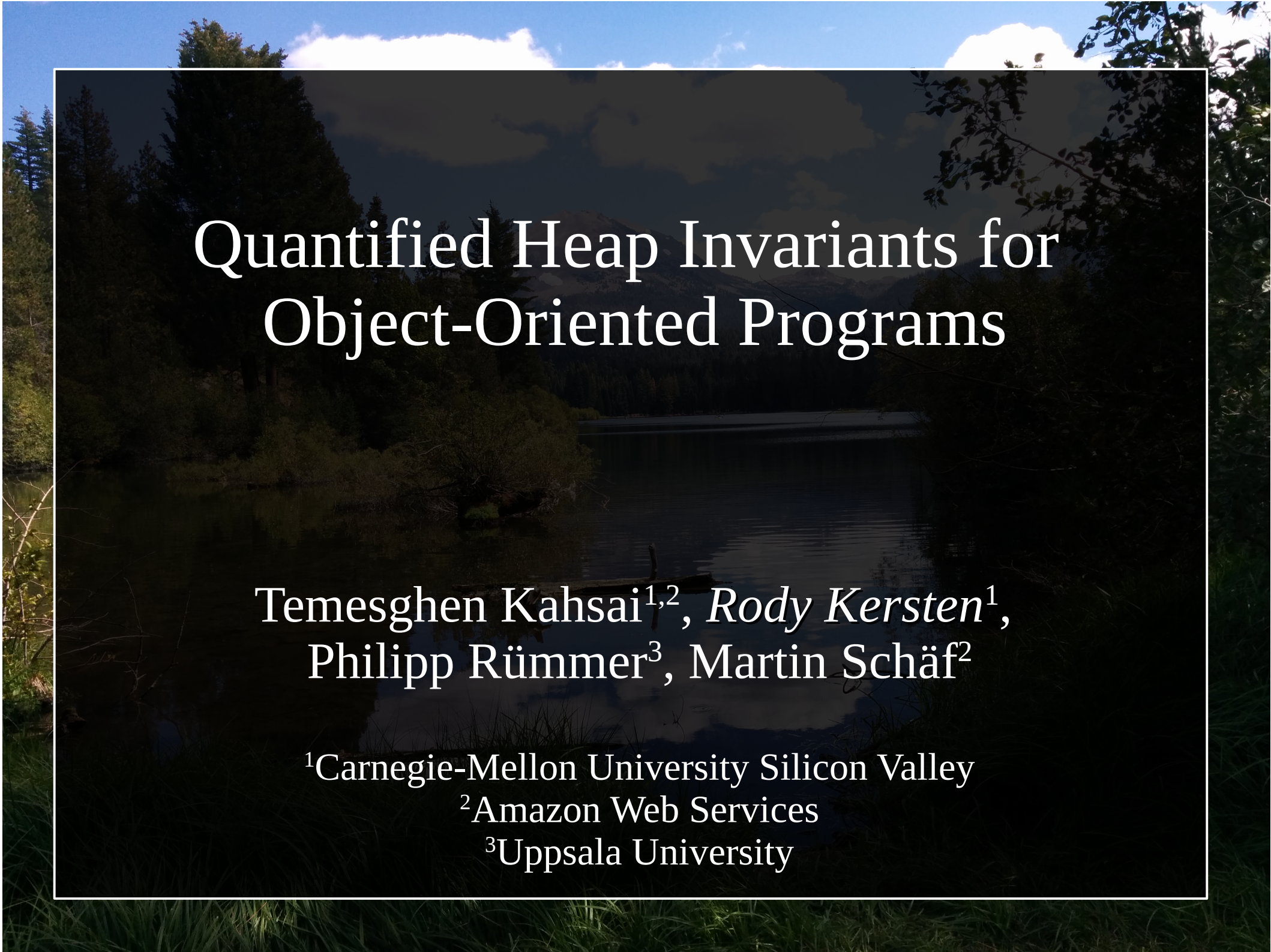


Quantified Heap Invariants for Object-Oriented Programs

Temesghen Kahsai[1,2], *Rody Kersten*[1], Philipp Rümmer[3], Martin Schäf[2]

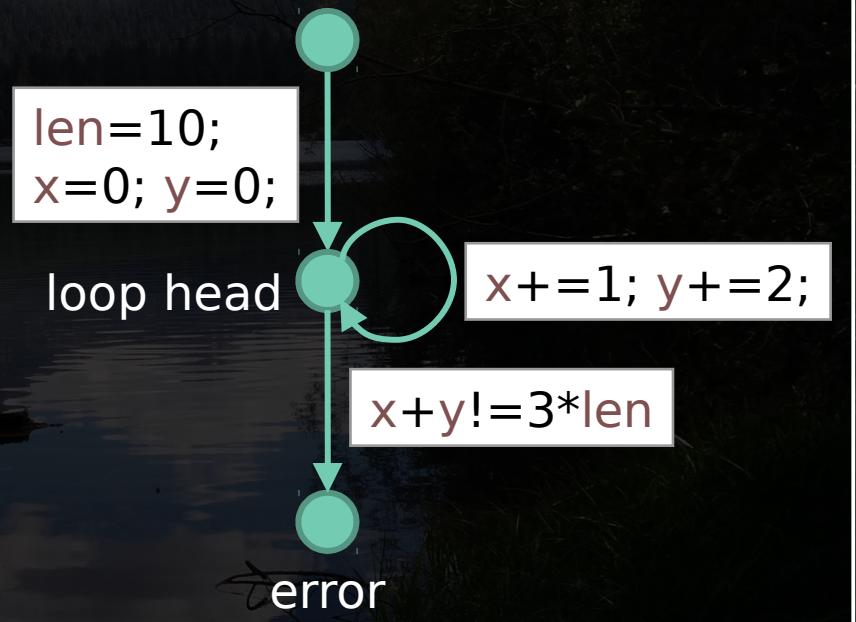
[1]Carnegie-Mellon University Silicon Valley
[2]Amazon Web Services
[3]Uppsala University

Verifying Java Programs

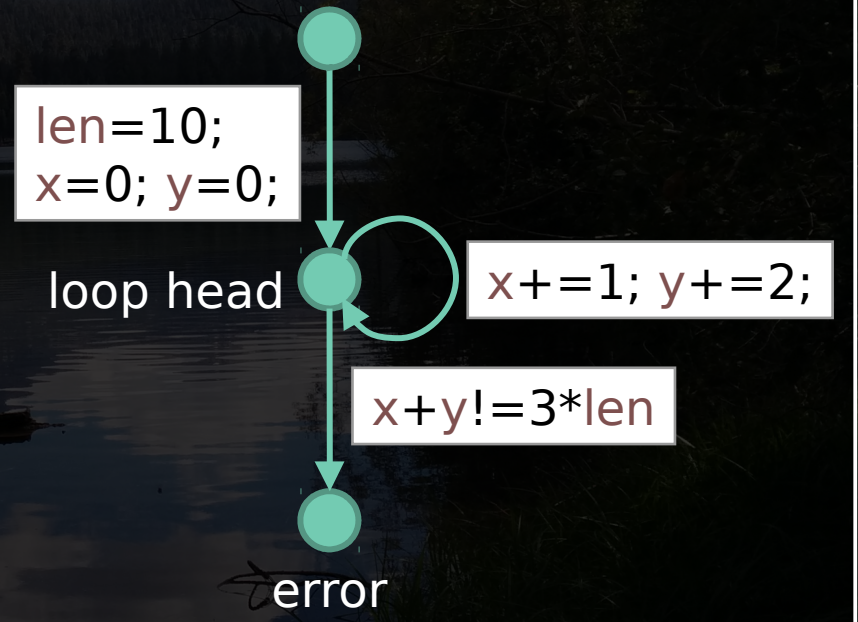
```
public class MyExample {
  public static void main(String[] args) {
    final int len=10;
    int x=0, y=0;
    while (x < len) {
      x+=1; y+=2;
    }
    assert x+y==3*len;
  }
}
```

Verifying Java Programs

```
public class MyExample {
  public static void main(String[] args) {
    final int len=10;
    int x=0, y=0;
    while (x < len) {
      x+=1; y+=2;
    }
    assert x+y==3*len;
  }
}
```



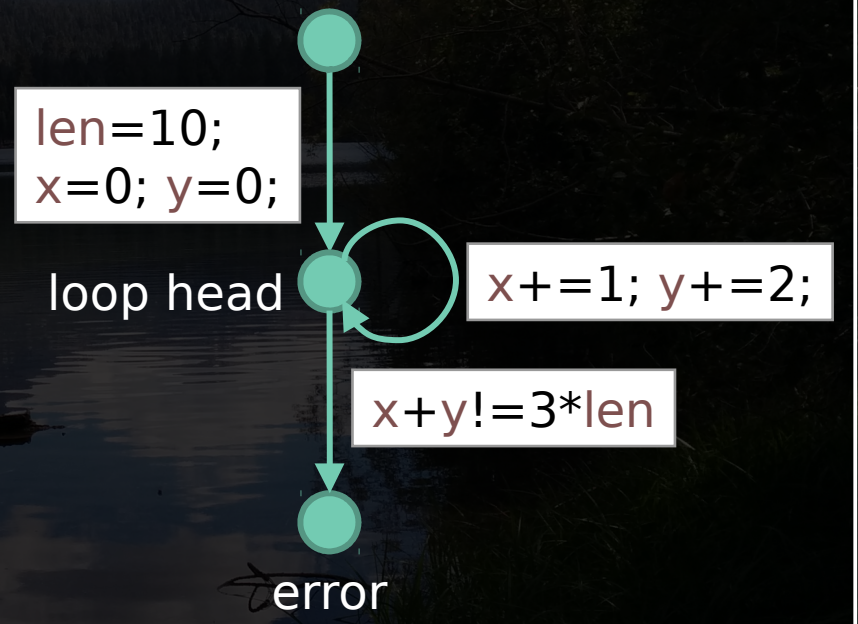
Verifying Java with Horn clauses



Verifying Java with Horn clauses

entry(x,y,len) ← true

head(0,0,10) ← entry(x,y,len)



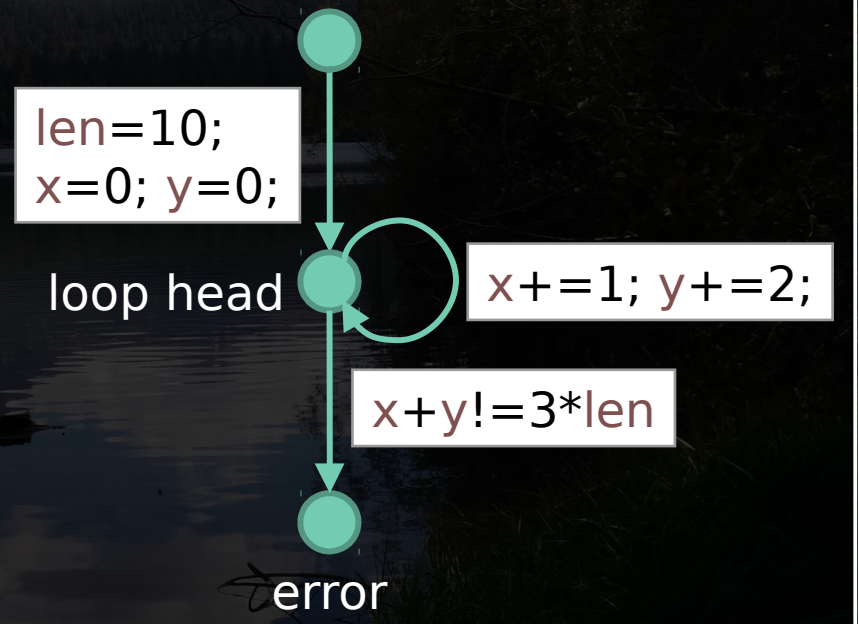
Verifying Java with Horn clauses

$$\text{entry}(x,y,len) \leftarrow \text{true}$$

$$\text{head}(0,0,10) \leftarrow \text{entry}(x,y,len)$$

$$\text{head}(x+1,y+2,len) \leftarrow x<len \wedge \text{head}(x,y,len)$$

$$3*len=x+y \leftarrow x>=len \wedge \text{head}(x,y,len)$$



Solving Horn clauses

$$\text{entry}(x,y,len) \leftarrow \text{true}$$

$$\text{head}(0,0,10) \leftarrow \text{entry}(x,y,len)$$

$$\text{head}(x+1,y+2,len) \leftarrow x<len \wedge \text{head}(x,y,len)$$

$$3*len=x+y \leftarrow x>=len \wedge \text{head}(x,y,len)$$

Use off-the-shelf solver

- Spacer (Z3)
- Eldarica
- …

Solving Horn clauses

$$\text{entry}(x,y,len) \leftarrow \text{true}$$

$$\text{head}(0,0,10) \leftarrow \text{entry}(x,y,len)$$

$$\text{head}(x+1,y+2,len) \leftarrow x<len \wedge \text{head}(x,y,len)$$

$$3*len=x+y \leftarrow x>=len \wedge \text{head}(x,y,len)$$

Predicate assignments:

entry(x,y,len)	?
head(x,y,len)	?

Solving Horn clauses

$$\text{entry}(x,y,len) \leftarrow \text{true}$$

$$\text{head}(0,0,10) \leftarrow \text{entry}(x,y,len)$$

$$\text{head}(x+1,y+2,len) \leftarrow x<len \wedge \text{head}(x,y,len)$$

$$3*len=x+y \leftarrow x>=len \wedge \text{head}(x,y,len)$$

Predicate assignments:

entry(x,y,len)	true
head(x,y,len)	3*len=x+y

Solving Horn clauses

$$\text{true} \leftarrow \text{true}$$

$$3*10=0+0 \leftarrow \text{true}$$

$$3*len=x+1+y+2 \leftarrow x<len \wedge (3*len=x+y)$$

$$3*len=x+y \leftarrow x>=len \wedge (3*len=x+y)$$

Predicate assignments:

entry(x,y,len)	true
head(x,y,len)	3*len=x+y

Solving Horn clauses

$$\text{entry}(x,y,len) \leftarrow \text{true}$$

$$\text{head}(0,0,10) \leftarrow \text{entry}(x,y,len)$$

$$\text{head}(x+1,y+2,len) \leftarrow x<len \wedge \text{head}(x,y,len)$$

$$3*len=x+y \leftarrow x>=len \wedge \text{head}(x,y,len)$$

Predicate assignments:

entry(x,y,len)	true
head(x,y,len)	y=2*x

Solving Horn clauses

$$\text{true} \leftarrow \text{true}$$

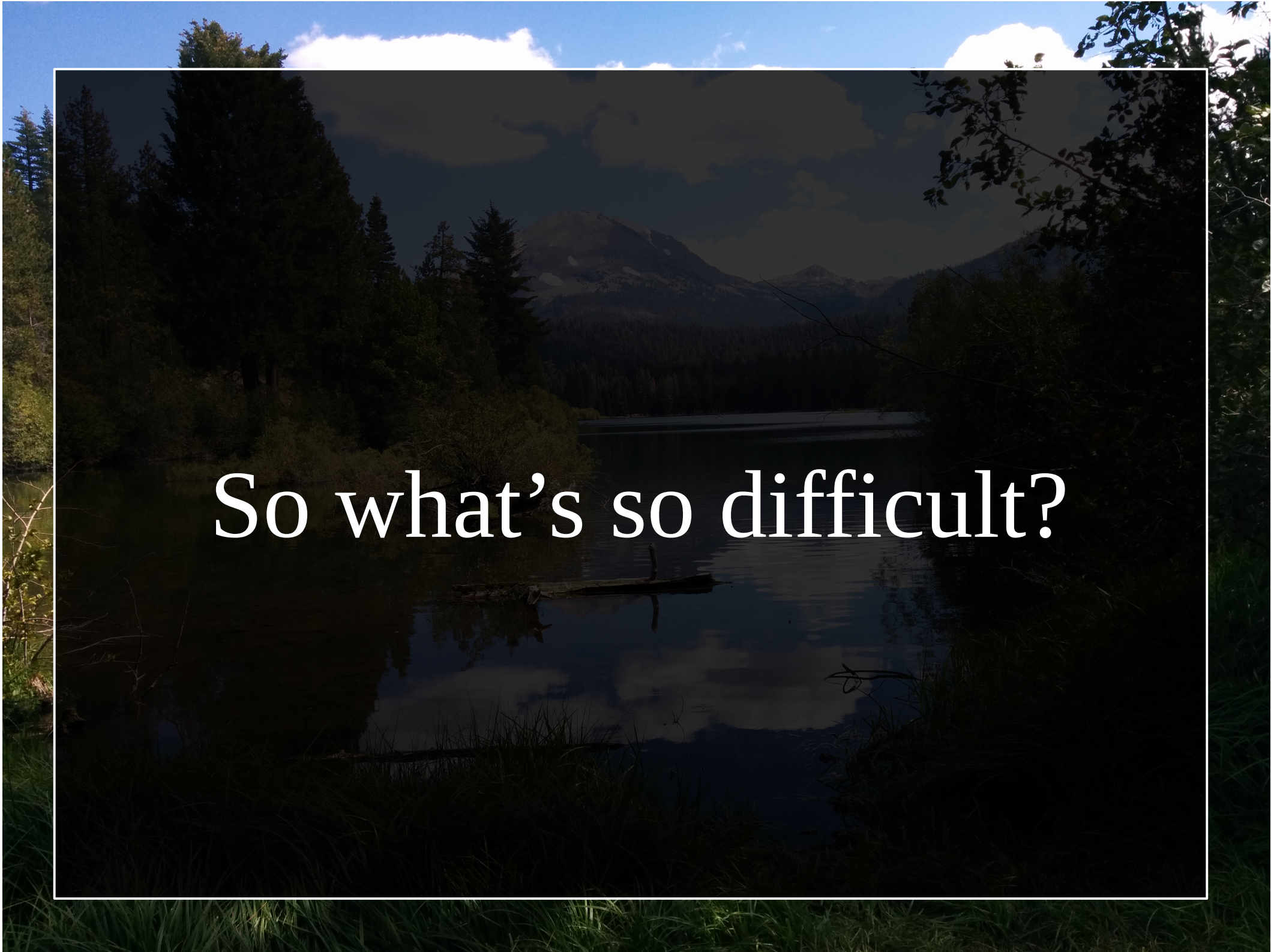
$$0=2*0 \leftarrow \text{true}$$

$$y+2=2*(x+1) \leftarrow x<len \wedge (y=2*x)$$

$$3*len=x+y \leftarrow x>=len \wedge (3*len=x+y)$$

Predicate assignments:

entry(x,y,len)	true
head(x,y,len)	y=2*x


So what’s so difficult?

How do we encode heap?

```
public static void main(String[] args) {
  final int size = 10;
  final int[] table = new int[size];
  Node l1 = null, l2 = null;
  for (int i=0; i<args.length; i++) {
    int d = Integer.parseInt(args[i]);
    if (d >= 0 && d < size)
      l1 = new Node(l1, d);
    else l2 = new Node(l2, d);
  }
  while (l1 != null) {
    table[l1.data]++;
    l1 = l1.next;
  }
}
```

```
public static class Node {
  final Node next;
  final int data;
  public Node(Node next, int data) {
    this.next = next;
    this.data = data;
  }
}
```

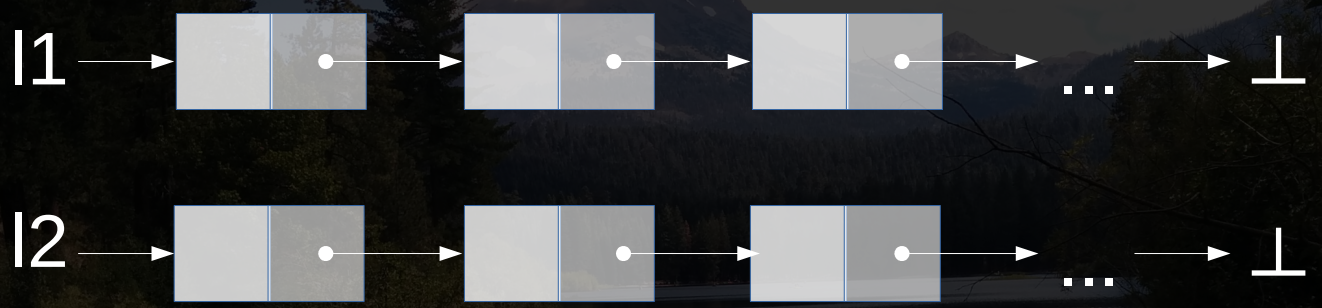
How do we encode heap?

```
public static void main(String[] args) {
  final int size = 10;
  final int[] table = new int[size];
  Node l1 = null, l2 = null;
  for (int i=0; i<args.length; i++) {
    int d = Integer.parseInt(args[i]);
    if (d >= 0 && d < size)
      l1 = new Node(l1, d);
    else l2 = new Node(l2, d);
  }
  while (l1 != null) {
    table[l1.data]++;
    l1 = l1.next;
  }
}
```

```
public static class Node {
  final Node next;
  final int data;
  public Node(Node next, int data) {
    this.next = next;
    this.data = data;
  }
}
```

For all Node objects reachable from l1 at this location: 0 <= l1.data < table.length

Space invariants



- Lightweight notion of invariants
- Summarize states of all objects of a particular class
- Specific to program and properties to be proven
- Can be inferred effectively using Horn solvers

Java to IVL

```
while (l1 != null) {
  table[l1.data]++;
  l1 = l1.next;
}
```

```
while l1 != null
  data, next = pull(l1)
  assert 0<=data<10
  tmp = pull(table, data)
  push(table, data, tmp+1)
  l1 = next
```

Removing heap interaction

```
while l1 != null
  data, next = pull(l1)
  assert 0<=data<10
  tmp = pull(table, data)
  push(table, data, tmp+1)
  l1 = next
```

```
while l1 != null
  havoc(data, next)
  assume Φ_Node(l1, data, next)
  assert 0<=data<10
  havoc(tmp)
  assume Φ_int[](table, data, tmp)
  assert Φ_int[](table, data, tmp+1)
  l1 = next
```

IVL to Horn clauses

assume $\Phi_{int[]}$(table, data, tmp)
assert $\Phi_{int[]}$(table, data, tmp+1)

p3(l1, table, next, tmp) ← $\Phi_{int[]}$(table, data, tmp) ∧ p2(l1, table, next)

$\Phi_{int[]}$(table, data, tmp+1) ← p3(l1, table, next, tmp)

Solve Horn clauses

```
public static void main(String[] args) {
  final int size = 10;
  final int[] table = new int[size];
  Node l1 = null, l2 = null;
  for (int i=0; i<args.length; i++) {
    int d = Integer.parseInt(args[i]);
    if (d >= 0 && d < size)
      l1 = new Node(l1, d);
    else l2 = new Node(l2, d);
  }
  while (l1 != null) {
    table[l1.data]++;
    l1 = l1.next;
  }
}
```

References as tuples

- Capure distinguishing features about reference
  - e.g. dynamic type, allocation site, values of final fields
- Immutable
- Reading/writing through pull/push would defeat their purpose
- e.g. nodes in l1 have different allocation site than nodes in l2

Solve Horn clauses

```
public static void main(String[] args) {
  final int size = 10;
  final int[] table = new int[size];
  Node l1 = null, l2 = null;
  for (int i=0; i<args.length; i++) {
    int d = Integer.parseInt(args[i]);
    if (d >= 0 && d < size)
      l1 = new Node(l1, d);
    else l2 = new Node(l2, d);
  }
  while (l1 != null) {
    table[l1.data]++;
    l1 = l1.next;
  }
}
```

Can be verified with these space invariants:

$$\phi_{\texttt{int[]}}(\texttt{o}, \texttt{idx}, \texttt{data}) := \mathit{true}$$

$$\phi_{\texttt{Node}}(\texttt{o}, \texttt{data}, \texttt{next}) := \texttt{o}{\downarrow_{\texttt{allocSite}}} = \texttt{l19} \rightarrow \big(0 \leq \texttt{data} < 10 \wedge (\texttt{next}{\downarrow_{\texttt{id}}} = 0 \vee \texttt{next}{\downarrow_{\texttt{allocSite}}} = \texttt{l19})\big)$$

Optimizations and extensions

- Optimized push-pull placement
- Method in-lining
- Flow sensitive space invariants
- Array invariants: include accessed index in space invariant

Limitations

- Space invariants can only express relation between multiple objects in special cases
- Ability to distinguish different objects depends on the precision of the static analysis

Implementation

- Available on GitHub
- Front-end Soot: Java bytecode → Jimple → Horn.
- Implementation is sound only for subset of Java. Threads, reflection, etc are not supported; integers are treated as natural numbers, etc.
- Supports different back-ends (Spacer and Eldarica)

Experimental results

		Correct		Incorrect		TO	N/A
		Safe	Unsafe	Imprecise	Unsound		
JayHorn benchmarks (41 safe, 41 unsafe)	JayHorn (Eldarica)	33	41	3	0	5	0
	JayHorn (Spacer)	37	41	3	0	1	0
	CPAChecker	5	5	0	0	0	72
MinePump (43 safe, 21 unsafe)	JayHorn (Eldarica)	32	21	11	0	0	0
	JayHorn (Spacer)	32	19	8	0	5	0
	CPAChecker	43	21	0	0	0	0
CBMC benchmarks (35 safe, 7 unsafe)	JayHorn (Eldarica)	29	7	3	0	0	3
	JayHorn (Spacer)	29	7	3	0	0	3
	CPAChecker	6	0	0	0	0	36
Total (119 safe, 76 unsafe)	JayHorn (Eldarica)	94	69	17	0	5	3
	JayHorn (Spacer)	98	67	14	0	6	3
	CPAChecker	54	26	0	0	0	108

Download, read development blog and papers, contribute!

http://jayhorn.github.io/jayhorn/