Datalog-based Scalable Semantic Diffing of Concurrent Programs

Chungha Sung | Shuvendu K. Lahiri | Constantin Enea
Chao Wang
Concurrent Programs
Evolving Software

becoming better

Fixing bugs or Adding features

Fixing bugs or Adding features

Fixing bugs or Adding features
Evolving Software

Fixing bugs or Adding features
Fixing bugs or Adding features
Fixing bugs or Adding features
Unexpected Behavior
Thread 1
lock(a);
x = 1;
y = x;
unlock(a);

Thread 2
lock(a);
\textcolor{red}{x} = 0;
unlock(a);
Thread 1

lock(a);

x = 1;

y = x;

unlock(a);

Thread 2

lock(a);

x = 0;

unlock(a);

New Read-from edge is created!!
Comparison after a change

Is there any unexpected new behavior?

NO!
Semantic difference

New data-flow edge
Prior work

• Bounded Model Checking (BMC) based approach
  - Need to instrument code with assertions
  - Interleaving enumeration => expensive

[Bouajjani et al. SAS 2017]
Our approach

• Constraint-based scalable program analysis
  
  - No code instrumentation needed
  
  - No interleaving enumeration
  
  - 10x to 1000x faster
  
  - Practically accurate
Outline

- Motivation

- Contribution
  *(Scalable approximate semantic diffing)*

- Experiments

- Conclusion
Overview

Datalog inference rules for semantic diffing

Scalable & Practically Accurate!

Compare the allowed data-flow edges over two programs
Overview

LLVM pass

$P_1$

$P_2$

Patch info

Datalog Facts

Datalog Facts

$\mu$Z Datalog Engine in Z3

Query

Differences

$\Delta_{12} = P_1^+ \setminus P_2^+$

$\Delta_{21} = P_2^+ \setminus P_1^+$

Semantic Differencing framework

Datalog Rules
Example

Thread1() {
    t = 0;
    x = 1;
    create(Thread2);
    lock(a);
    ...
    assert(x != t);
    unlock(a);
}

Thread2() {
    lock(a);
    t = x;
    ...
    x = 2;
    unlock(a);
}
Example

Thread1() {
    t = 0;
x = 1;
create(Thread2);
lock(a);
...
assert(x != t);
unlock(a);
}

Thread2() {
    lock(a);
t = x;
...
x = 2;
unlock(a);
}
Example

Thread1() {
  t = 0;
  x = 1;
  create(Thread2);
  lock(a);
  ...
  assert(x != t);
  unlock(a);
}

Thread2() {
  lock(a);
  t = x;
  ...
  x = 2;
  unlock(a);
}
Example

Thread1() {
    t = 0;
    x = 1;
    create(Thread2);
    lock(a);
    ...
    assert(x != t);
    unlock(a);
}

Thread2() {
    lock(a);
    t = x;
    ...
    x = 2;
    unlock(a);
}
Example

Thread1() {
    t = 0;
    x = 1;
    create(Thread2);
    lock(a);
    ...
    assert(x != t);
    unlock(a);
}

Thread2() {
    lock(a);
    t = x;
    ...
    x = 2;
    unlock(a);
}
Example

Thread1() {
    t = 0;
    x = 1;
    create(Thread2);
    lock(a);
    ...
    assert(x != t);
    unlock(a);
}

Thread2() {
    lock(a);
    t = x;
    ...
    x = 2;
    unlock(a);
}

Assertion is not violated
Example

Thread1() {
    t = 0;
    x = 1;
    create(Thread2);
    lock(a);
    ...
    assert(x != t);
    unlock(a);
}

Thread2() {
    lock(a);
    t = x;
    ...
    x = 2;
    unlock(a);
}

Assertion is not violated
Example after a change

Thread1() {
    t = 0;
    x = 1;
    create(Thread2);
    lock(a);
    ...
    assert(x != t);
    unlock(a);
}

Thread2() {
    lock(a);
    t = x;
    ...
    x = 2;
    unlock(a);
}
Example after a change

Thread1() {
    t = 0;
    x = 1;
    create(Thread2);
    lock(a);
    ...
    assert(x != t);
    unlock(a);
}

Thread2() {
    lock(a);
    t = x;
    ...
    x = 2;
    unlock(a);
}

Assertion is violated
Overview

\[ \Delta_{12} = P_1^+ \setminus P_2^+ \]
\[ \Delta_{21} = P_2^+ \setminus P_1^+ \]

Datalog Facts

Datalog Engine in Z3

Datalog Rules

Semantic Diffing framework

LLVM pass

Patch info

Query
Program Analysis in Datalog

Evolving concurrent programs $\rightarrow$ Datalog facts

Datalog Rules

Semantic difference checking between the two programs

[Whaley & Lam, 2004]
[Livshits & Lam, 2005]
What is Datalog?

• Declarative language for deductive database [Ullman 1989]

**Facts**
parent (bill, mary)
parent (mary, john)

**Rules**
ancestor (X, Y) ← parent (X, Y)
ancestor (X, Y) ← parent (X, Z), ancestor (Z, Y)

*New relationship: ancestor (bill, john)*
Datalog Translation

```plaintext
Thread1() {
    t = 0;
    1: x = 1;
    create(Thread2);
    lock(a);
    ...
    2: assert(x != t);
    unlock(a);
}

Thread2() {
    lock(a);
    3: t = x;
    ...
    4: x = 2;
    unlock(a);
    }
```

**MustHappenBefore relations**

po (s1, s2) -> MustHB (s1, s2)
ThreadOrder(s1, t1, s2, t2) -> MustHB(s1, s2)

**Inferred relations**

MustHB: (\{1, 2\}, \{3, 4\}, \{1, 3\}, \{1, 4\})
Datalog Translation

Thread1() {
    t = 0;
    1: x = 1;
    create(Thread2);
    lock(a);
    ...
    2: assert(x != t);
    unlock(a);
}

Thread2() {
    lock(a);
    3: t = x;
    ...
    4: x = 2;
    unlock(a);
}

MustHappenBefore relations
po (s1, s2) -> MustHB (s1, s2)
ThreadOrder(s1, t1, s2, t2) ->
MustHB(s1, s2)

Inferred relations
MustHB: {{1, 2}, {3, 4}, {1, 3}, {1, 4}}
## Datalog Translation

### Thread1()
```
t = 0;
1: x = 1;
   create(Thread2);
   lock(a);
   ...
2: assert(x != t);
   unlock(a);
}
```

### Thread2()
```
lock(a);
3: t = x;
   ...
4: x = 2;
   unlock(a);
}
```

### MayHappenBefore relations
- **MustHB** ($s_1, s_2$) --> **MayHB** ($s_1, s_2$)
- Not **ThreadOrder** ($s_1, t_1, s_2, t_2$) --> **MayHB**($s_2, s_1$)

### Inferred relations
- **MustHB**: ($\{1, 2\}$, $\{3, 4\}$, $\{1, 3\}$, $\{1, 4\}$)
- **MayHB**: ($\{1, 2\}$, $\{3, 4\}$, $\{1, 3\}$, $\{1, 4\}$, $\{2, 3\}$, $\{2, 4\}$, $\{3, 2\}$, $\{4, 2\}$)
Datalog Translation

Thread1() {
    t = 0;
    1: x = 1;
    create(Thread2);
    lock(a);
    ...
    2: assert(x != t);
    unlock(a);
}

Thread2() {
    lock(a);
    3: t = x;
    ...
    4: x = 2;
    unlock(a);
}

MayHappenBefore relations
MustHB (s1, s2) -> MayHB (s1, s2)

Not ThreadOrder(s1, t1, s2, t2) ->
MayHB(s2, s1)

Inferred relations
MustHB: ({1, 2}, {3, 4}, {1, 3}, {1, 4})
MayHB: ({1, 2}, {3, 4}, {1, 3}, {1, 4}, {2, 3}, {2, 4}, {3, 2}, {4, 2})
Datalog Translation

Thread1() {
  t = 0;
  1: x = 1;
  create(Thread2);
  lock(a);
  ...
  2: assert(x != t);
  unlock(a);
}

Thread2() {
  lock(a);
  3: t = x;
  ...
  4: x = 2;
  unlock(a);
}

MayReadFrom relations
MayHB (s1, s2) & St(s1) & Ld(s2) ->
MayRF (s1, s2)

Inferred relations
MustHB: ({1, 2}, {3, 4}, {1, 3}, {1, 4})
MayHB: ({1, 2}, {3, 4}, {1, 3}, {1, 4}, {2, 3}, {2, 4}, {3, 2}, {4, 2})
MayRF: ({1, 2}, {1, 3}, {3, 2}, {4, 2})
Datalog Translation

Thread1() {
    t = 0;
    1: x = 1;
    create(Thread2);
    lock(a);
    ...
    2: assert(x != t);
    unlock(a);
}

Thread2() {
    lock(a);
    3: t = x;
    ...
    4: x = 2;
    unlock(a);
}

Rank2 relations

CS

W(x)

R(x)

CS

W(x)

PostDom

R(x)
Datalog Translation

Thread1() {
    t = 0;
    1: x = 1;
    create(Thread2);
    lock(a);
    ...
    2: assert(x != t);
    unlock(a);
}

Thread2() {
    lock(a);
    3: t = x;
    ...
    4: x = 2;
    unlock(a);
}

Rank2 relations

CS

W(x)

R(x)

R(x)

W(x)

CS

RF1

RF2

RF3

PostDom
Datalog Translation

Thread1() {
    t = 0;
    1: x = 1;
    create(Thread2);
    lock(a);
    ...
    2: assert(x != t);
    unlock(a);
}

Thread2() {
    lock(a);
    3: t = x;
    ...
    4: x = 2;
    unlock(a);
}

Rank2 relations

RF1 -> not RF3
RF2 -> not RF1
Datalog Translation

Thread1() {
    t = 0;
    x = 1;
    create(Thread2);
    lock(a);
    ...
    2: assert(x != t);
    unlock(a);
}

Thread2() {
    lock(a);
    3: t = x;
    ...
    4: x = 2;
    unlock(a);
}

Inferred relations
MustHB: ({1, 2}, {3, 4}, {1, 3}, {1, 4})
MayHB: ({1, 2}, {3, 4}, {1, 3}, {1, 4}, {2, 3}, {2, 4}, {3, 2}, {4, 2})
MayRF: ({1, 2}, {1, 3}, {3, 2}, {4, 2})
Rank2: ([[{1, 2} -> {1, 3}], [{1, 3} -> {4, 2}]]

Rank2 relations
RF1 -> not RF3
RF2 -> not RF1
Datalog Translation

Thread1() {
    t = 0;
    1: x = 1;
    create(Thread2);
    lock(a);
    ...
    2: assert(x != t);
    unlock(a);
}

Thread2() {
    lock(a);
    3: t = x;
    ...
    4: x = 2;
    unlock(a);
}

Rank2 relations

W(x) R(x)
RF1 > not RF3
RF2 > not RF1

Inferred relations
MustHB: ({{1, 2}, {3, 4}, {1, 3}, {1, 4}})
MayHB: ({{1, 2}, {3, 4}, {1, 3}, {1, 4}, {2, 3}, {2, 4}, {3, 2}, {4, 2}})
MayRF: ({{1, 2}, {1, 3}, {3, 2}, {4, 2}})
Rank2: ({{1, 2} -> {1, 3}, {1, 3} -> {4, 2}, {1, 3} -> {1, 2}})
Overview

**LLVM pass**

$P_1$

$P_2$

Patch info

**Datalog Facts**

**μZ Datalog Engine in Z3**

**Query**

**Differences**

$\Delta_{12} = P_1^+ \setminus P_2^+$

$\Delta_{21} = P_2^+ \setminus P_1^+$

**Semantic Diffing framework**

**Semantic Diffing framework**
Computing differences

MayRF (s1, s2, p1) & Not MayRF(s1, s2, p2) -> DiffP1-P2 (s1, s2)
MayRF (s1, s2, p2) & Not MayRF(s1, s2, p1) -> DiffP2-P1 (s2, s1)
Computing differences

May be allowed in P1
(\([\{1, 2\} \rightarrow \{1, 3\}], [\{1, 3\} \rightarrow \{4, 2\}]\))

May be allowed in P2
(\([\{1, 2\} \rightarrow \{1, 3\}], [\{1, 3\} \rightarrow \{4, 2\}], [\{1, 3\} \rightarrow \{1, 2\}]\))
## Experimental Results 1

<table>
<thead>
<tr>
<th>The first set</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td># of apps</td>
<td>41</td>
</tr>
<tr>
<td>LOC</td>
<td>5,546</td>
</tr>
<tr>
<td>Types</td>
<td>Sync, Th.Order, St.Order, Cond</td>
</tr>
</tbody>
</table>

### Sources

[Bouajjani et al. *SAS 2017*]
[Yu & Narayanasamy *ISCA 2009*]
[Beyer *TACAS 2015*]
[Bloem et al. *FM 2014*]
[Lu et al. *ASPLOS 2008*]

[Herlihy & Shavit *The Art of Multiprocessor Programming 2008*]
[Open source bug reports]
Comparison

• Bounded Model Checking based approach

[Bouajjani et al. SAS 2017]
## Experimental Results 1

<table>
<thead>
<tr>
<th>The first set</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Execution time of BMC-based approach</td>
<td>&gt; 3 hours</td>
</tr>
<tr>
<td>Execution time of our approach (NEW)</td>
<td>15.57 seconds</td>
</tr>
<tr>
<td># of differences our approach found</td>
<td>402 dataflow edges (All valid)</td>
</tr>
</tbody>
</table>
## Experimental Results 2

<table>
<thead>
<tr>
<th>The second set</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td># of apps</td>
<td>6</td>
</tr>
<tr>
<td>LOC</td>
<td>7,986</td>
</tr>
<tr>
<td>Types</td>
<td>Th.Order, Cond</td>
</tr>
<tr>
<td>Sources</td>
<td>[Yang et al. <em>U. of Utah 2008</em>] [Yu &amp; Narayanasamy <em>ISCA 2009</em>]</td>
</tr>
<tr>
<td>BMC-based approach</td>
<td><strong>Not available</strong></td>
</tr>
<tr>
<td>Execution time of our approach</td>
<td><strong>140.28 seconds</strong></td>
</tr>
<tr>
<td># of differences our approach found</td>
<td><strong>72 (All valid)</strong></td>
</tr>
</tbody>
</table>
Conclusions

• Proposed a *Datalog based* static analysis for semantic diffing concurrent programs

• *Practically accurate* for identifying differences in thread synchronization

• Significant improvement in *scalability* especially for large programs
Thank you!

https://github.com/chunghasung/EC-Diff