IC637 Program Analysis

Lecture 6: Context Sensitivity

Minseok Jeon

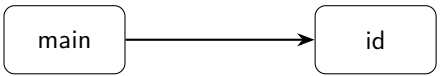
2025 Fall

Necessity of Context Sensitivity

- Without context sensitivity, it is unable to prove the castings are always safe.

```
1  class A{}
2  class B{}
3  class C{
4    static Object id(Object v) {
5      return v;
6    }
7    void main(String[] args) {
8      A a = new A();//l1
9      B b = new B();//l2
10     A a2 = (A)id(a);//query1
11     B b2 = (B)id(b);//query2
12   }
13 }
```

Analysis results (context insensitive):



$$a \to \{l_1\}\ b \to \{l_2\}\ v \to \{l_1, l_2\}$$
$$a2 \to \{l_1, l_2\}\ b2 \to \{l_1, l_2\}$$

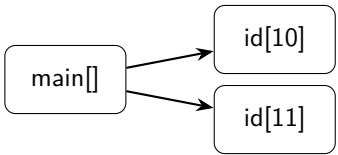
- How can we prove the castings are always safe?

Necessity of Context Sensitivity

- By analyzing the method id context sensitively, we can prove the queries.

```
1  class A{}
2  class B{}
3  class C{
4    static Object id(Object v) {
5      return v;
6    }
7    void main(String[] args) {
8      A a = new A();//l1
9      B b = new B();//l2
10     A a2 = (A)id(a);//query1
11     B b2 = (B)id(b);//query2
12   }
13 }
```

Analysis results (context sensitive):



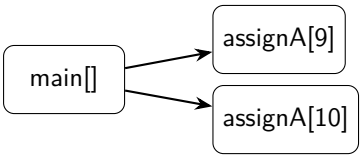
$$a[] \to \{l_1\}\ b[] \to \{l_2\}$$
$$v[10] \to \{l_1\}\ v[11] \to \{l_2\}$$
$$a2[] \to \{l_1\}\ b2[] \to \{l_2\}$$

Necessity of Context Sensitive Heap

- Describe analysis results for the following example.

```
1  class A {}
2
3  public class Tmp {
4    static A assignA() {
5      A v = new A();//l1
6      return v;
7    }
8    void main(String[] args) {
9      A a1 = assignA();
10     A a2 = assignA();
11     assert (a1 != a2);//query
12   }
13 }
```

Analysis results (without heap context):



Call graph of context sensitive analysis

$$v[9] \to \{l_1\}\ v[10] \to \{l_1\}$$
$$a1[] \to \{l_1\}\ a2[] \to \{l_1\}$$

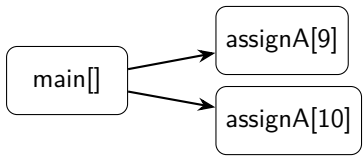
- How can we prove the assertion?

Context Sensitive Heap

- For precision, heaps should be also analyzed in a context sensitive manner.

```
1  class A {}
2
3  public class Tmp {
4    static A assignA() {
5      A v = new A();//l1
6      return v;
7    }
8    void main(String[] args) {
9      A a1 = assignA();
10     A a2 = assignA();
11     assert (a1 != a2);//query
12   }
13 }
```

Analysis results (context sensitive):



Call graph of context sensitive analysis

$$v[9] \rightarrow \{l_1[9]\}\ v[10] \rightarrow \{l_1[10]\}$$
$$a1[] \rightarrow \{l_1[9]\}\ a2[] \rightarrow \{l_1[10]\}$$

Context Sensitive Analysis

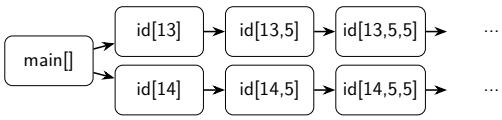
- Give an example showing context sensitive analysis does not terminate.

Necessity of Context Abstraction

- Full context sensitive analysis does not terminate when it analyzes:

```
1  class A{} class B{}
2  class C{
3    Object id(int i, Object v) {
4      if (i > 0) {
5          return id(i-1,v);
6      } else {
7          return v;}
8    }
9    void main(String[] args) {
10     int i = input();
11     A a = new A();//l1
12     B b = new B();//l2
13     A a2 = (A)id(i,a);//query1
14     B b2 = (B)id(i,b);//query2
15   }
16 }
```

Analysis results (context sensitive):



Call graph of context sensitive analysis

$$a[] \to \{l_1\}\ b[] \to \{l_2\}$$
$$v[13] \to \{l_1\}\ v[14] \to \{l_2\}$$
$$v[13,5] \to \{l_1\}\ v[14,5] \to \{l_2\}$$
$$v[13,5,5] \to \{l_1\}\ v[14,5,5] \to \{l_2\}$$
$$\dots$$

Necessity of Context Abstraction

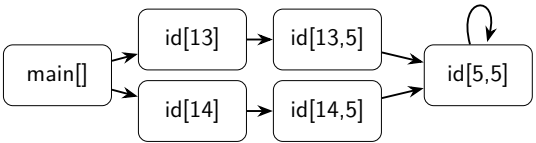
- How can we make the context sensitive analysis always terminate?

(Last) K-limited Context Sensitive Analysis

- Conventional k-context sensitive analysis keeps last-k context elements.

```
1  class A{} class B{}
2  class C{
3    Object id(int i, Object v) {
4      if (i > 0) {
5          return id(i-1,v);
6      } else {
7          return v;}
8    }
9    void main(String[] args) {
10     int i = input();
11     A a = new A();//l1
12     B b = new B();//l2
13     A a2 = (A)id(i,a);//query1
14     B b2 = (B)id(i,b);//query2
15   }
16 }
```

Analysis results:



Call graph of (last) 2-context sensitivity

$$a[] \to \{l_1\}\ b[] \to \{l_2\}$$
$$v[13] \to \{l_1\}\ v[14] \to \{l_2\}$$
$$v[13,5] \to \{l_1\}\ v[14,5] \to \{l_2\}$$
$$v[5,5] \to \{l_1, l_2\}$$
$$a2[] \to \{l_1, l_2\}\ b2[] \to \{l_1, l_2\}$$

Input Relations

$Alloc(var : V, heap : H)$

$Move(to : V, from : V)$

$Load(to : V, base : V, fld : F)$

$Store(base : V, fld : F, from : V)$

$FormalParam(method : M, arg : A, var : V)$

$FormalReturn(method : M, var : V)$

$ActualParam(invocation : I, arg : A, var : V)$

$ActualReturn(invocation : I, var : V)$

$ThisVar(meth : M, this : V)$

$HeapType(heap : H, type : T)$

$LookUp(type : T, sig : S, methM)$

$VirtualCall(base : V, sig : S, invo : I, inMeth : M)$

V : the set of program variables

H : the set of heap locations

F : the set of fields

M : the set of method identifiers

S : the set of method signatures (names)

I : the set of instructions

T : the set of class types

$\mathbb{N}$: the set of natural numbers

C : the set of contexts

HC : the set of heap contexts

Output Relations

$$VarPointsTo(var : V, heap : H)$$
$$FldPointsTo(baseH : H, fld : F, heap : H)$$
$$CallGraph(invo : I, meth : M)$$

$$\downarrow$$

$$VarPointsTo(var : V, ctx : C, heap : H, heapCtx : HC)$$
$$FldPointsTo(baseH : H, heapCtx : HC, fld : F, heap : H, heapCtx : HC)$$
$$CallGraph(invo : I, callerCtx : C, meth : M, calleeCtx : C)$$

Example of Context Sensitive Analysis

```
class A{}
class B{}
class C{
  Object id(Object v) {
    return v;
  }
  void main(String[] args) {
    A a = new A();//l1
    B b = new B();//l2
    A a2 = (A)id(a);//l3, query1
    B b2 = (B)id(b);//l4, query2
  }
}
```

VarPointsTo(a, [], l_1, [])
VarPointsTo(b, [], l_2, [])
CallGraph(l_3, [], id, [l_3])
CallGraph(l_4, [], id, [l_4])
VarPointsTo(v, [l_3], l_1, [])
VarPointsTo(v, [l_4], l_2, [])
VarPointsTo(a2, [], l_1, [])
VarPointsTo(b2, [], l_2, [])

Example of Context Sensitive Analysis

```
class A {}

public class Tmp {
  static A assignA() {
    return new A(); //l1
  }
  void main(String[] args) {
    A a1 = assignA();//l2
    A a2 = assignA();//l3
    assert (a1 != a2);//query
  }
}
```

VarPointsTo(a1, [], l1, [l2])
VarPointsTo(a2, [], l1, [l3])
CallGraph(l2, [], assignA, [l2])
CallGraph(l3, [], assignA, [l3])

Context Constructors

- Different choices of constructors yield different context sensitivity flavors

$$\textbf{Record}(heap : H, ctx : C) = newHCtx : HC$$
$$\textbf{Merge}(heap : H, hctx : HC, invo : I, ctx : C) = newCtx : C$$

- **Record** generates heap contexts
- **Merge** generates calling contexts

Analysis Rules

(1) $\textbf{Record}(heap, ctx) = hctx,$

$VarPointsTo(var, ctx, heap, hctx) \leftarrow$

$CallGraph(\_, \_, meth, ctx), Alloc(var, heap, meth)$

(2) $VarPointsTo(var, ctx, heap, hctx) \leftarrow$

$CallGraph(\_, \_, meth, ctx), Move(to, from, meth),$

$VarPointsTo(from, ctx, heap, hctx)$

Analysis Rules

(3) $FldPointsTo(baseH, baseHCtx, fld, heap, hctx) \leftarrow$
$Store(base, fld, from, meth), VarPointsTo(from, ctx, heap, hctx),$
$VarPointsTo(base, ctx, baseH, baseHCtx), CallGraph(\_, \_, meth, ctx)$

(4) $VarPointsTo(to, ctx, heap, hctx) \leftarrow$
$Load(to, base, fld), VarPointsTo(base, ctx, baseH, baseHCtx),$
$FldPointsTo(baseH, baseHCtx, fld, heap, hctx), CallGraph(\_, \_, meth, ctx)$

Analysis Rules

(5) $\textbf{Merge}(heap, hctx, invo, callerCtx) = calleeCtx,$

$VarPointsTo(this, calleeCtx, heap, hctx),$

$CallGraph(invo, callerCtx, toMeth, calleeCtx) \leftarrow$

$VCall(base, sig, invo, inMeth), CallGraph(\_, \_, inMeth, callerCtx),$

$VarPointsTo(base, callerCtx, heap, hctx),$

$HeapType(heap, heapT), LookUp(heapT, sig, toMeth),$

$ThisVar(toMeth, this)$

Analysis Rules

(6) $VarPointsTo(param', calleeCtx, heap, hctx) \leftarrow$
$CallGraph(invo, callerCtx, meth, calleeCtx),$
$FormalParam(meth, n, param', param), ActualParam(invo, n, param),$
$VarPointsTo(param, callerCtx, heap, hctx)$

(7) $VarPointsTo(return, callerCtx, heap, hctx) \leftarrow$
$CallGraph(invo, callerCtx, meth, calleeCtx),$
$FormalReturn(meth, return), ActualReturn(invo, return),$
$VarPointsTo(return, calleeCtx, heap, hctx)$

Record and Merge for 1-Context Sensitivity

- **Record** and **Merge** for standard 1-context sensitivity, so-called 1-call-site sensitivity (e.g., $C : \mathbb{I}, HC : \mathbb{I}$), is as follows:

$$\textbf{Record}(heap, ctx) = ctx$$

$$\textbf{Merge}(heap, hctx, invo, callerCtx) = invo$$

Open Question

- Define suitable **Record** and **Merge** that perform well in practice.

Exercise

- Define **Record** and **Merge** for 1-context sensitivity that prove the queries.

```
1  class A{} class B{}
2  class C{
3    Object id(Object v) {
4      return v;}
5  }
6  public class D {
7    void main(String[] args) {
8      A a = new A();//l1
9      B b = new B();//l2
10     C c = new C();//l3
11     A a2 = (A)c.id(a);//query1
12     B b2 = (B)c.id(b);//query2
13   }
14 }
```

Exercise

- Define **Record** and **Merge** for 2-context sensitivity that prove the queries.

```
1  class A{} class B{}
2  class C{
3    Object id(Object v) {
4      return v;}
5    Object id1(Object v) {
6      return id(v);}
7  }
8  public class D {
9    void main(String[] args) {
10     A a = new A();//l1
11     B b = new B();//l2
12     C c = new C();//l3
13     A a2 = (A)c.id(a);//query1
14     B b2 = (B)c.id(b);//query2
15   }
16 }
```

Quiz

- Can we prove the queries with k-context sensitive analysis?

```
1  class A{} class B{}
2  class C{
3    Object id(int i, Object v) {
4      if (i > 0) {
5          return id(i-1,v);
6      } else {
7          return v;}}}
8  public class D {
9    void main(String[] args) {
10     int i = input();
11     A a = new A();//l1
12     B b = new B();//l2
13     C c = new C();//l3
14     A a2 = (A)c.id(i,a);//query1
15     B b2 = (B)c.id(i,b);//query2
16   }}
```

Quiz

- Define **Record** and **Merge** for 1-context sensitivity that prove the queries.

```
1  class A{} class B{}
2  class C{
3    Object id(int i, Object v) {
4      if (i > 0) {
5          return id(i-1,v);
6      } else {
7          return v;}}}
8  public class D {
9    void main(String[] args) {
10     int i = input();
11     A a = new A();//l1
12     B b = new B();//l2
13     C c = new C();//l3
14     A a2 = (A)c.id(i,a);//query1
15     B b2 = (B)c.id(i,b);//query2
16   }}
```

Wrap-up: Context Sensitivity

- **Context Insensitive Analysis**: Merges all calling contexts, leading to imprecision
  - Cannot distinguish different call sites
- **Full Context Sensitivity**: Distinguishes all calling contexts
  - May not terminate (infinite contexts in recursive calls)
- **K-limited Context Sensitivity**: Abstracts contexts to ensure termination
  - Keeps up to k context elements