IC637 Program Analysis

Lecture 7: Context Depth and Flavors

Minseok Jeon

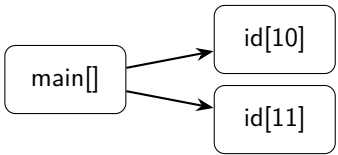
2025 Fall

Review: Necessity of Context Sensitive Analysis

- 1-**call-site sensitive** analysis can prove the castings are always safe.
- call-site sensitivity uses the call sites as context elements.

```
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 }
```

1-**call-site** sensitive analysis



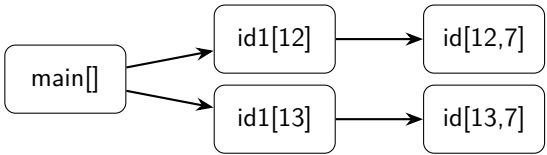
$$a[] \rightarrow \{l_1[]\}\ b[] \rightarrow \{l_2[]\}$$
$$v[10] \rightarrow \{l_1[]\}\ v[11] \rightarrow \{l_2[]\}$$
$$a2[] \rightarrow \{l_1[]\}\ b2[] \rightarrow \{l_2[]\}$$

2-Call-Site Sensitive Analysis

- conventional 2-**call-site sensitive** uses the caller methods' contexts.

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

2-**call-site** sensitive analysis



$$a[] \rightarrow \{l_1[]\}\ b[] \rightarrow \{l_2[]\}$$
$$id1.v[12] \rightarrow \{l_1[]\}\ id1.v[13] \rightarrow \{l_2[]\}$$
$$id.v[12,7] \rightarrow \{l_1[]\}\ id.v[13,7] \rightarrow \{l_2[]\}$$
$$a2[] \rightarrow \{l_1[]\}\ b2[] \rightarrow \{l_2[]\}$$

Call-Site Sensitive Analysis

- Define **Record** and **Merge** for 1-call-site-sensitivity + 0-context-sensitive heap.

Record(heap, ctx) =
Merge(heap, hctx, invo, callerCtx) =

- Define **Record** and **Merge** for 1-call-site-sensitivity + 1-context-sensitive heap.

Record(heap, ctx) =
Merge(heap, hctx, invo, callerCtx) =

- Define **Record** and **Merge** for 2-call-site-sensitivity + 1-context-sensitive heap.

Record(heap, ctx) =
Merge(heap, hctx, invo, callerCtx) =

- Define **Record** and **Merge** for 2-call-site-sensitivity + 2-context-sensitive heap.

Record(heap, ctx) =
Merge(heap, hctx, invo, callerCtx) =

Quiz

- How can we prove the queries using conventional 1-context sensitive analysis?

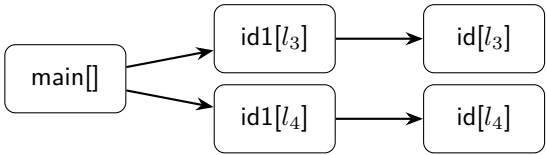
```
1  class A{} class B{}
2  class C{
3    Object id(Object v) {
4      return v;
5    }
6    Object id1(Object v) {
7      return this.id(v);
8    }
9  }
10 class D{
11   void main(String[] args) {
12     A a = new A();//l1
13     B b = new B();//l2
14     C c1 = new C();//l3
15     C c2 = new C();//l4
16     A a3 = (A)c1.id1(a);//query3
17     B b3 = (B)c2.id1(b);//query4
18   }
19 }
```

Object Sensitivity

- Object sensitivity uses the receiver objects as context elements.

```
1  class A{} class B{}
2  class C{
3    Object id(Object v) {
4      return v;
5    }
6    Object id1(Object v) {
7      return this.id(v);
8    }
9  }
10 class D{
11   void main(String[] args) {
12     A a = new A();//l1
13     B b = new B();//l2
14     C c1 = new C();//l3
15     C c2 = new C();//l4
16     A a3 = (A)c1.id1(a);//query3
17     B b3 = (B)c2.id1(b);//query4
18   }
19 }
```

1-**object** sensitive analysis



$$a[] \rightarrow \{l_1[]\}\ b[] \rightarrow \{l_2[]\}$$
$$id1.v[l_3] \rightarrow \{l_1[]\}\ id1.v[l_4] \rightarrow \{l_2[]\}$$
$$id.v[l_3] \rightarrow \{l_1[]\}\ id.v[l_4] \rightarrow \{l_2[]\}$$
$$a2[] \rightarrow \{l_1[]\}\ b2[] \rightarrow \{l_2[]\}$$

2-Object Sensitivity

- How can we define 2-object sensitivity that can prove the queries?

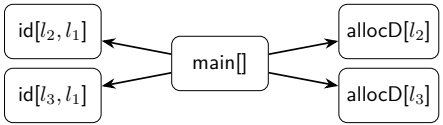
```
1  class A{} class B{}
2  class C{
3    D allocD() {
4      return new D();}//l1
5  }
6  class D{
7    Object id(Object v) {
8      return v;}
9  }
10 class E{
11   void main(String[] args) {
12     C c1 = new C();//l2
13     C c2 = new C();//l3
14     D d1 = c1.allocD();
15     D d2 = c2.allocD();
16     A a = (A)d1.id(new A());//query1
17     B b = (B)d2.id(new B());//query2
18 }}
```

2-Object Sensitivity

- 2-**object sensitive** analysis uses the heap contexts of the receiver objects.

```
1  class A{} class B{}
2  class C{
3    D allocD() {
4      return new D();}//l1
5  }
6  class D{
7    Object id(Object v) {
8      return v;}
9  }
10 class E{
11   void main(String[] args) {
12     C c1 = new C();//l2
13     C c2 = new C();//l3
14     D d1 = c1.allocD();
15     D d2 = c2.allocD();
16     A a = (A)d1.id(new A());//l4
17     B b = (B)d2.id(new B());//l5
18 }}
```

2-**object** sensitive analysis



$$c1[] \rightarrow \{l_2[]\}\ c2[] \rightarrow \{l_3[]\}$$
$$d1[] \rightarrow \{l_1[l_2]\}\ d2[] \rightarrow \{l_1[l_3]\}$$
$$v[l_2, l_1] \rightarrow \{l_4[]\}\ v[l_3, l_1] \rightarrow \{l_5[]\}$$
$$a[] \rightarrow \{l_4[]\}\ b[] \rightarrow \{l_5[]\}$$

2-Object Sensitivity

- Typical code pattern that needs 2-object sensitivity

```
class A{}
class B{}
class C {
  public static void main (){
  ArrayList al1 = new ArrayList();//AL1
  ArrayList al2 = new ArrayList();//AL2

  al1.add(new A());
  al2.add(new B());

  ArrayList.ListItr it1 = al1.iterator();
  ArrayList.ListItr it2 = al2.iterator();

  A a = (A)it1.next(); //Query 1
  B b = (B)it2.next(); //Query 2
  }
}
```

```
class ArrayList{
  Object[] elementData = new Object[10];
  int size = 0;
  void add(Object e){
    elementData[size++] = e;
  }
  ListItr iterator(){
    return new ListItr(); //IT
  }
  class ListItr{
    int cursor = 0;
    Object next(){
      return elementData[cursor++];
    }
  }
}
```

Object Sensitivity

- Define **Record** and **Merge** for 1-object-sensitivity + 0-context-sensitive heap.

Record(heap, ctx) =
Merge(heap, hctx, invo, callerCtx) =

- Define **Record** and **Merge** for 1-object-sensitivity + 1-context-sensitive heap.

Record(heap, ctx) =
Merge(heap, hctx, invo, callerCtx) =

- Define **Record** and **Merge** for 2-object-sensitivity + 1-context-sensitive heap.

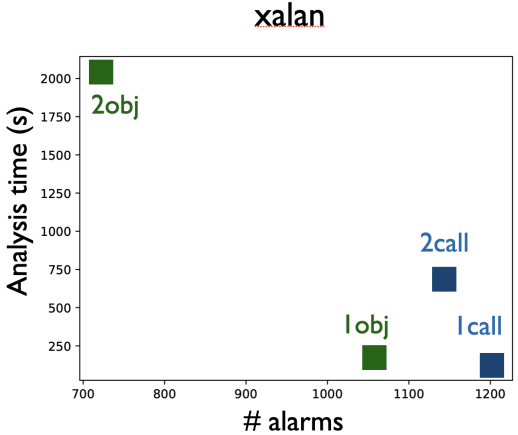
Record(heap, ctx) =
Merge(heap, hctx, invo, callerCtx) =

- Define **Record** and **Merge** for 2-object-sensitivity + 2-context-sensitive heap.

Record(heap, ctx) =
Merge(heap, hctx, invo, callerCtx) =

2-Object Sensitivity

- 2-object sensitivity is known to be highly precise but also very expensive analysis.



Quiz

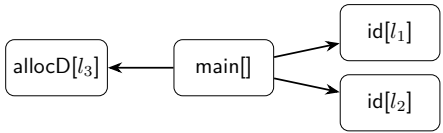
- (2-)object sensitivity is precise but very expensive.
- Try to define a new context flavor which is a faster (but less precise) version of object sensitivity.

Problem of Object Sensitivity

- Problem of object sensitivity in terms of scalability.

```
1  class C{
2    D allocD() {
3      if (cond()){
4        return new D();//l1
5      }
6      else{
7        return new D();//l2
8      }
9    }
10 }
11 class D{
12   Object id(Object v) {
13     return v;}
14 }
15 class E{
16   void main(String[] args) {
17     C c = new C();//l3
18     D d = c.allocD();
19     A a = (A)d.id(new A());//l4
20   }
21 }
```

1-**object** sensitive analysis



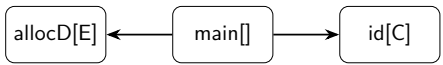
$$c[] \rightarrow \{l_3[]\}$$
$$d[] \rightarrow \{l_1[], l_2[]\}$$
$$v[l_1] \rightarrow \{l_4[]\}$$
$$v[l_2] \rightarrow \{l_4[]\}$$

Type Sensitivity

- Type sensitivity[1] is a coarser version of object sensitivity.
- Type sensitivity uses class types instead of objects.

```
1  class C{
2    D allocD() {
3      if (cond()){
4        return new D();//l1
5      }
6      else{
7        return new D();//l2
8      }
9    }
10 }
11 class D{
12   Object id(Object v) {
13     return v;}
14 }
15 class E{
16   void main(String[] args) {
17     C c = new C();//l3
18     D d = c.allocD();
19     A a = (A)d.id(new A());//l4
20   }
21 }
```

1-**object** sensitive analysis



$$c[] \rightarrow \{l_3[]\}$$
$$d[] \rightarrow \{l_1[], l_2[]\}$$
$$v[C] \rightarrow \{l_4[]\}$$

[1] https://dl.acm.org/doi/10.1145/1925844.1926390

Type Sensitivity

- Define **Record** and **Merge** for 1-type-sensitivity + 0-context-sensitive heap where *toClass*: $\mathbb{H} \to \mathbb{C}$ maps each heap allocation site to its allocating class (e.g., $\mathit{toClass}(l_1) = \mathsf{C}$).

Record(heap, ctx) =
Merge(heap, hctx, invo, callerCtx) =

- Define **Record** and **Merge** for 1-type-sensitivity + 1-context-sensitive heap.

Record(heap, ctx) =
Merge(heap, hctx, invo, callerCtx) =

- Define **Record** and **Merge** for 2-type-sensitivity + 1-context-sensitive heap.

Record(heap, ctx) =
Merge(heap, hctx, invo, callerCtx) =

- Define **Record** and **Merge** for 2-type-sensitivity + 2-context-sensitive heap.

Record(heap, ctx) =
Merge(heap, hctx, invo, callerCtx) =

Quiz

- Try to define a new context flavor that is a coarser version of call-site sensitivity.

Quiz

- Try to define a new context flavor that is a coarser than object sensitivity but more precise than type sensitivity.

Quiz

- When $i < j$, if i-call-site sensitivity proves a query, does j-call-site sensitivity also prove the same query? If so, why? If not, give a counterexample.

- When $i > j$, if i-call-site sensitivity proves a query, does j-call-site sensitivity also prove the same query? If so, why? If not, give a counterexample.

Quiz

- If i-call-site sensitivity proves a query, does j-object sensitivity also prove the same query? If so, why? If not, give a counterexample.

- If i-object sensitivity proves a query, does j-call-site sensitivity also prove the same query? If so, why? If not, give a counterexample.

Quiz

- If k-type sensitivity proves a query, does k-object sensitivity also prove the same query? If so, why? If not, give a counterexample.

- If k-object sensitivity proves a query, does k-type sensitivity also prove the same query? If so, why? If not, give a counterexample.

Quiz

- When $i < j$, if i-object sensitivity proves a query, does j-type sensitivity also prove the same query? If so, why? If not, give a counterexample.