Analysis Sensitivity

Types of *Context*

Call strings presume that the *location* of a call-site (or sequence of call-sites) is relevant to the analysis.

Assumption sets presume that the *data* at a call-site, i.e., data state at the point of the call, is relevant to the analysis.

Which do you think is more the broadly applicable notion of context?

The following examples illustrate the value of assumption sets.

Small Assumption Sets for Signs Analysis

```
public int foo(int x) {
  if (x > 0) return 0;
  else return 1;
}
```

How can assumption sets impact precision in calculating $\mathcal{A}_{signs}[\![foo()]\!]$, i.e., the return value?

Analysis Sensitivity

The Value of Small Assumption Sets

Without an assumption set

$$\{+, 0\}$$

With an assumption set of $Signs$ for the single parameter x

$$\{(+, 0), (-, +), (0, +)\}$$

using $Signs \times Signs$ as the context-sensitive lattice.

This allows us to distinguish the two different return values

The Limitations of Small Assumption Sets

```
public int bar(int x, int y) {
  if (x > 0) return -1;
  else if (y > 0) return 1;
  else return 0;
}
```

With an assumption set of $Signs$ for the single parameter x

$$\{(+,-),(-,\{0,+\}),(0,\{0,+\})\}$$

if we use $Signs \times \mathcal{P}(Signs)$, or (even worse)

$$\{(+,-),(-,0),(-,+),(0,0),(0,+)\}$$

if we use $Signs \times Signs$

Analysis Sensitivity

Large Assumption Sets for Signs

```
public int bar(int x, int y) {
  if (x > 0) return -1;
  else if (y > 0) return 1;
  else return 0;
}
```

With an assumption set of $Signs \times Signs$ for parameters x and y

$$\{((+,+),-),((+,0),-),((+,1),-),((-,+),+),((-,0),0),$$
$$((-,-),0),((0,+),+),((0,0),0),((0,-),0)\}$$

for the context-sensitive lattice $(Signs \times Signs) \times Signs$

Lots of precision, but large sets to flow around.

Analysis Sensitivity

Refining the Precision of Analysis Results

Do we really care who calls us?

```
public int main(...) {
    ...
    foo(2);
    ...
    foo(2);
    ...
}
```

A method cannot tell who called it only what values were passed, i.e., the *calling context*.

Refining the Precision of Analysis Results

What really matters is whether the differences in calling context are relevant from the point of view of our analysis.

```
public int main(...) {
   ...
   foo(3);
   ...
   foo(2);
   ...
}
```

```
public void bat(...) {
    foo(2);
}

public int baz(...) {
    return foo(3);
}
```

Analysis Sensitivity

The *Art* of Static Analysis

If cost was no object we would perform an MVP analysis

- Not only is cost an object, but it blows up exponentially

The art of program analysis is designing techniques to select *relevant* features of IVPs to include in the analysis

- where relevant means that the feature could possibly influences analysis results

An analysis that captures such relevant features is said to be *sensitive*

Analysis Sensitivity

Sensitivity in Static Analysis

Flow-insensitive

The order in which statements execute is ignored.

Flow-sensitive

The control flow order in which statements may execute in a sequential program is taken into account.

(partial) Path-sensitive

The influence of data in determining the order of statement execution is taken into account.

Analysis Sensitivity

Context-sensitive

Information about method calling context is taken into account.

Object-sensitive

An assumption-set-like approach to context sensitivity that uses the *value* of the receiver object as the context.

Thread-sensitive

The control flow order in which statements execute in a multi-threaded program is considered; subsumes flow-sensitive.

Synchronization-sensitive

The influence of *lock* state in determining the order of statement execution in a multi-threaded program is considered; a special case of path-sensitivity layered on thread-sensitivity.

The mechanism for adding sensitivity is a lattice like $\mathcal{P}(\Delta \times D)$

Analysis Sensitivity

You Be The Judge

What kind of sensitivity do we need for state propagation analysis?

What kind of sensitivity do we need to determine whether a method may write to a global variable?

What kind of sensitivity do we need for points-to analysis?

Analysis Sensitivity

Sensitivity is a Tradeoff (sometimes it's essential)

Different forms of sensitivity can be combined and applied to the same problem

- flow-insensitive, context-insensitive points-to analysis
- flow-insensitive, context-sensitive points-to analysis
- flow-sensitive, context-insensitive points-to analysis
- flow-sensitive, object-sensitive points-to analysis
- flow-sensitive, parameterized points-to analysis

with or without *field sensitivity*.

We'll see concrete examples of each of these combinations.

Analysis Sensitivity

Flow-insensitive points-to

```
        void foo(A a) {
[0]       A x;
[1]       ...
[2]       x = a;
[3]       ...
[4]       x = a.next;
[5]       ...
        }
```

Consider every assignment as *adding* information

$$A_{in}(l_1) = A_{in}(l_3) = A_{in}(l_5) = \{(x, \{[\![null]\!], [\![a]\!], [\![a.next]\!]\})\}$$

where, for example, $[\![a]\!]$ denotes a set of heap locations containing all actual instances that can be refered to by a on some program execution.

Analysis Sensitivity

Flow-sensitive points-to

```
       void foo(A a) {
[0]      A x;
[1]      ...
[2]      x = a;
[3]      ...
[4]      x = a.next;
[5]      ...
       }
```

Consider every assignment as *adding* and *updating* information

$$
\begin{aligned}
A_{in}(l_1) &= \{(x, \{[\![null]\!]\})\} \\
A_{in}(l_3) &= \{(x, \{[\![a]\!]\})\} \\
A_{in}(l_5) &= \{(x, \{[\![a.next]\!]\})\}
\end{aligned}
$$

Analysis Sensitivity

Flow-sensitive vs. Flow-insensitive points-to

Flow-sensitivity may provide additional precision and it may cost more to calculate.

calculation of fix-point over statements vs. linear scan of statements

The **key difference** is the ability to perform *strong updates* (i.e., kills)

Flow values may be quite complex, e.g., graph representations of abstractions of the heap.

In practice, this distinction is mitigated by the use of *static single assignment* (SSA) which converts a class of flow-sensitive problems into flow-insensitive problems with little loss in precision.

Analysis Sensitivity

Context-sensitive points-to

```
       void foo(A a) {             void bar() {
[0]      A x;                        A y = new A();
[1]      ...                         y.next = new A();
[2]      x = a;                      foo(y);
[3]      ...                         y.next = y;
[4]      x = a.next;                 foo(y);
[5]      ...                       }
       }
```

Distinguishing call sites `bar.2` and `bar.4` may be beneficial

$$A_{in}(l_{bar.2})(\text{y.next}) \neq A_{in}(l_{bar.4})(\text{y.next})$$

Could use either location or assumption set to distinguish the contexts in this case, but not in general.

Analysis Sensitivity

Sensitivity Varies

As we've seen it can vary

- with the nature of the analysis problem
- with our tradeoff between cost and precision
- across different parts of the program!

Consider state propagation analysis for the $(open; close)^*$

where `open()` and `close()` are application methods

How should we analyze library code?