CSSE4630 Week 7 Lab: Interprocedural Analysis

Mark Utting

Version 1.0

1 Introduction

This workshop is focussed on Interprocedural Analysis — analysing function calls and returns. No coding this week, just using existing interprocedural analyses, to compare how well they work.

In the second part of this workshop, you can also start work on your assignment and ask questions about it.

2 Evaluating Interprocedural Analysis

Use interprocedural analysis with the propagation solver to analyse the `examples/signs_fun.tip` program. You must add the `-normalizecalls` flag as well, to split each function call node into separate Call and AfterCall nodes.

```
tip -normalizecalls -sign iwlrp examples/signs_fun.tip
```

You should get sensible results, with each different copy of the function (`idf(a)`, `posf(a)` and `negf(a)`) having different 'sign' inputs and outputs. But notice how the `idf(a)` function is analysed as having unknown inputs and outputs (Top), because it is called once with a positive input and once with a negative input. We will really like to get better analysis results than this.

And in a real program we would want to write just a single function rather than these three identical copies of the identity function.

So try out the `signs3.tip` program (see Blackboard, Week 7, Workshop Exercise) to see how this interprocedural analysis handles a more realistic program with function calls. You should see that the function input is classified as *Top* (unknown sign), so all the function results are also *Top*. This is not very accurate.

```
tip -sign iwlrp examples/signs3.tip
```

TIP has four different interprocedural worklist solvers that can work with the Sign analysis:

- **iwlr:** worklist solver with reachability, interprocedural version [not implemented yet]
- **iwlrp:** worklist solver with reachability and propagation, interprocedural version
- **csiwlrp:** worklist solver with reachability and propagation, context-sensitive (with call string) interprocedural version
- **cfiwlrp:** worklist solver with reachability and propagation, context-sensitive (with functional approach) interprocedural version

You have already tried the `iwlrp` solver, which does not consider the caller context, so just gives one very approximate analysis result that is applicable to ALL the callers.

Let's try the last two solvers and see if they give better results? You will need to rename the output graph each time, to save it so that you can compare the graphs.

```
tip -sign csiwlrp examples/signs3.tip
tip -sign cfiwlrp examples/signs3.tip
```

You should see the the *call string* analysis (`csiwlrp`) does a separate analysis for each call of the function, so does the *Pos* analysis twice! But the *functional* analysis approach uses *parameter sensitivity* to recognise that the function has already been analysed with a *Pos* input, so does not need to be analysed again.

In the worst case, these context sensitive interprocedural analyses can be much less efficient than the simple content-insensitive analysis (`iwlrp` solver). But they do give much more accurate results for programs like this one.

3 Working on your Static Analysis Assignment

Please use the rest of this workshop to work on your assignment, and ask your tutor any questions that you have about it.