Usage of Data Stored in Map Data Structures

Bachelor's Thesis Project Description

Lowis Engel
Supervised by Dr. Caterina Urban and Jérôme Dohrau

ETH Zürich

March 2018

1 Introduction

With the ever increasing amount of available digital data, analyzing it is getting more and more relevant. This led to the rise of data science for decision making in many fields. However, as the dependence on the decisions of these applications grows, errors can have increasingly severe consequences, especially in areas like economy [2] or medicine.

So it is important to ensure correctness of these programs. In that regard non-fatal programming errors, where there is no clear indication that something went wrong, can be particularly dangerous. One type of these errors, which is relevant especially in data science, is entirely unused input data, which may still lead to plausible results. Here we define input data to be used, if there is a dependency between the input and the outcome of a program (directly or indirectly via conditions or assignments). This means it is used, if the outcome of the program can be influenced by different concrete inputs.

Consider the following Python program[1] as an example for this kind of errors, which takes a dictionary of texts indexed by the authors as input and should output the (weighted) number of occurences for each word in all texts (so-called 'bag-of-words'). Due to 'bug A' the program is not counting the words in the texts but in the authors. 'Bug B' causes the program to execute the word counting only for texts not in the 'important' authors set.

```
1  from collections import defaultdict
2  important = {''Albert Einstein'', ''Alan Turing''}
3  texts = dictinput()        #{''<author>'' : ''<text>''}
4
5  freqdict = defaultdict(int)    #initialized to 0
6  for a,b in texts.items():
7      if (a in important):       #texts of important authors weighted twice
8          weight = 2
9      else:
10         weight = 1
11         words = a.split()       #Bug A: Should be 'b' (values)
12         for word in words:      #and Bug B: Wrong indentation
13             word = word.lower()
14             freqdict[word] += weight
15 print(freqdict)          #outputs <word>:<count>, ...
```

Example 1

Part of the Lyra project[2] is aiming at detecting such errors in (simplified) programs written in Python, one of the most popular programming languages in data science. It uses static analysis by abstract interpretation [1] to automatically infer unused input data of a given program. This means it is iterating over the program's statements (in backwards order) producing an overapproximation of the used inputs in an abstract usage domain as its fixpoint, without executing the program on concrete inputs. This approach leads to a sound result, in the sense that every input labeled as used in the

[1] cf. https://www.datacamp.com/community/tutorials/python-dictionary-tutorial
[2] http://www.pm.inf.ethz.ch/research/lyra.html

abstract domain may be used by the concrete program and inputs labeled as unused are definitely not used (see Figure 1).

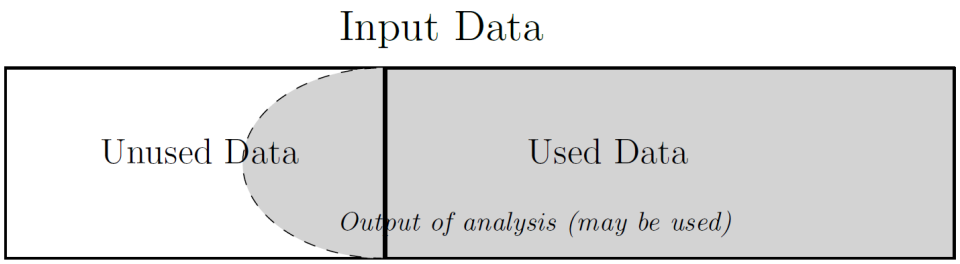

Figure 1: In grey: Overapproximation of the used data as given by our analysis

Lyra already includes this analysis for single inputs and input lists of statically unknown length [5]. The goal of this thesis is to extend the analysis theoretically and implementation-wise, such that we are able to analyze dictionaries[3] (map data structures). This other important data structure of Python is an unordered set of key-value pairs, which means there is no inherent ordering of the elements as it is the case for lists. This is one reason, why new abstractions must be found.

2 Core Goals

(with a rough estimate of the required time on the right side)

- Find examples for the usage of dictionaries in data science programs and the associated possible errors. These could be manually created or deducted from real applications, data science courses or coding competitions. Real-world examples may be hard to find, because their source code is often not published. That is why we also look at coding competitions, which are not necessarily data science related, but provide us with a large set of simple benchmarks. These examples will help us to get a better undestanding of the problems we are trying to detect. (**)
- Design an extension of the overapproximating static analysis of the Lyra project for analyzing usage of Python dictionaries within the framework of abstract interpretation. This includes defining abstact domain objects for dictionaries (possibly abstracting keys and values independently) and extending the simplified Python language. We approach this problem step by step, increasing the complexity of the analysis:
  - Abstract dictionaries as a single object. This means, a dictionary will be used, if any of its element is used. In Example 1 this would mean, that the analysis will output that the whole dictionary 'texts' is not used in line 3, if we consider a dictionary only used, if some values are used and not only the keys. This could detect 'bug A'. If we resolve 'bug A', this analysis could not detect 'bug B' and would have the result, that the *whole* dictionary is used. (*)
  - Apply a strongly live variable analysis [3], an extension of the live variable analysis used by compilers to detect usage before initialization by assignment. An abstracted part of a dictionary will be strongly live, if it is used in an assignment to another strongly live object or in a statement other than an assignment. Here an abstract state is a set of strongly live objects. Assuming we associate keys and values with each other in our abstraction and have a key abstraction that can capture conditions on strings like in line 7 of our Example 1, this could possibly also recognize 'bug B' in the case that 'bug A' has been resolved. More specifically it may detect, that only the values of keys not in the set 'important' (which is statically known) are used, if we use a fine-grained dictionary abstraction. (**)
  - To also capture implicit information flows (like objects used in an if-condition, in whose branch another object is used) use the more precise data usage analysis introduced in [4](section 10). That is, having four levels of usage (*used, not-used, below, overwritten*) to capture the usage at different nesting levels. This analysis still works purely syntactically. For our Example 1 this could only improve the usage analysis of the keys. Because it would possibly not declare all of them as used, but only the ones used in the else-branch, since the 'weight' in the if-branch is never used and so the 'a' in the condition does not become used. So in a sense this analysis would detect both bugs at the same time (assuming the same abstraction as above). (***)

[3] `https://docs.python.org/3/library/stdtypes.html#typesmapping`

- Implement the designed analyses for Lyra (in Python). This includes implementing the abstract domain for dictionaries and defining semantics for transforming the abstract states while traversing the control flow graph (for dictionary operations). (∗∗∗)
- Evaluate the analyses on the found Examples. This especially includes measurement of running time and precision. To assess the precision define what errors should be found and check if they are actually found. (∗)

3 Possible Extensions

(with a rough estimate of the required time on the right side)

- Integrate the analysis as a plugin into PyCharm using the IntelliJ Platform SDK to make it easier to apply for real-world applications. The plugin could for example highlight unused variables in the code or give some other graphical feedback. (∗∗)
- Make the results more precise by combining the overapproximating analysis with an underapproximating analysis (see Figure 2). By that we can specify objects that are definitely used and reduce the fraction of objects that *may* be used. The underapproximating analysis could work with (simple) numerical abstract domains to see if different inputs in that domain cause different abstractions for the outputs, which would mean that the input definitely has an effect on the outputs. For example one could use the *even-odd* domain on a simple program that just adds one to the input. Then an even (odd) input would lead to an odd (even) output, which would mean that the input is definitely used. As opposed to the above analysis this is an semantic approach.
  The results could also be even further improved by using testing to reduce the 'false-negatives', inputs which are actually used, but not recognized by this static analysis (see Figure 2) (∗∗∗)
- Make the analysis inter-procedural, as right now it works (mostly) intra-procedural. This would make it more usable for real-world programs. (∗∗)
- Let the analysis exclude respectively not report intentionally unused input data, which could be automatically inferred or explicitely defined by the user. For example an underscore (_) in Python could be a good indication. In Example 1 there could for instance be an underscore instead of 'a' in line 6 (without the if-conditional) or the texts of some authors could be intentionally excluded and the user could specify that. (∗∗)
- Extend the analysis to include more complex language constructs, such as nested dictionaries (and lists), which are used, when multiple keys are needed. (∗∗)

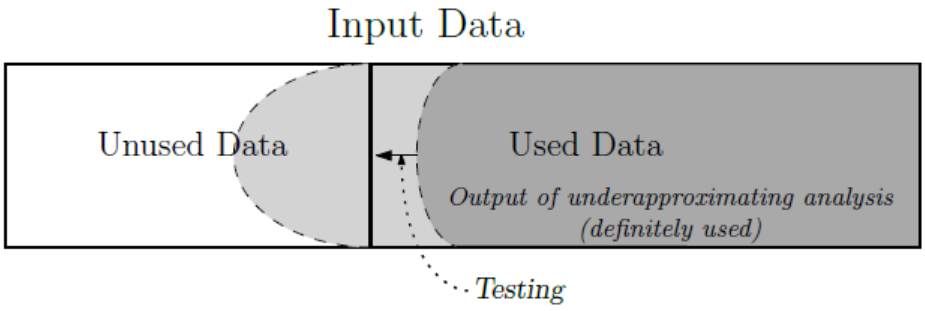

Figure 2: In dark grey: Underapproximation of the used data as given by the additinal analysis

References

[1] P. Cousot and R. Cousot. "Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints". In: *POPL*. ACM Press, 1977, pp. 238–252.

[2] Thomas Herndon, Michael Ash, and Robert Pollin. "Does high public debt consistently stifle economic growth? A critique of Reinhart and Rogoff". In: *Cambridge Journal of Economics* 38.2 (2014), pp. 257–279. URL: `https://EconPapers.repec.org/RePEc:oup:cambje:v:38:y:2014:i:2:p:257-279.`.

[3] R. Wilhelm R. Giegerich U. Möncke. "Invariance of Approximative Semantics with Respect to Program Transformations". In: *GI — 11. Jahrestagung*. Springer Berlin Heidelberg, 1981, pp. 1–10. URL: `https://doi.org/10.1007/978-3-662-01089-1_1`.

[4] C. Urban and P. Müller. "An Abstract Interpretation Framework for Input Data Usage". In: *European Symposium on Programming (ESOP)*. LNCS. To appear. Springer-Verlag, 2018.

[5] S. Wehrli. "Static Program Analysis of Data Usage Properties". MA thesis. ETH Zurich, Zurich, Switzerland, 2017.