Inferring Counter-Examples from Abstract Error States via Backward Analysis

Master's Thesis Problem Description

Raphael Fuchs
Supervisor: Lucas Brutschy

Background

The TouchDevelop programming environment [6, 8] enables hobbyist programmers to create scripts for their smartphones. Applications may be written on the devices themselves, and they have access to all things that matter in a mobile environment: music and images on the device, sensors, shared data in the cloud and social networks.

Since users of TouchDevelop are usually not experienced programmers, a lot of published scripts contain errors. Furthermore, the environment lacks traditional rigorous testing facilities. The TouchBoost project attempts to increase the reliability of scripts and detect possible errors by employing static analysis techniques. It utilizes the Sample framework (Static Analysis of Multiple LanguagEs), which is a generic static analyzer based on the abstract interpretation theory [2, 3], developed at the Chair of Programming Methodology at ETH. Sample supports and already contains a wide range of different analyses [4, 9, 1].

When an analyzer like TouchBoost finds an abstract state with a potential error, the analysis usually stops there and the problem is reported. However, this is not very helpful to a TouchDevelop user, since the reported problem may not be easy to understand. A problem is often only triggered with a particular set of inputs. There is also the possibility of false alarms because the analysis is incomplete and necessarily leads to an over-approximation of program behaviours.

Core Task

The main goal of this thesis is to improve the situation when a possible violation of some property is found by the analyzer: We try to infer a counter-example, that is, a concrete execution leading to the error. If we can find one, we provide the user with concrete inputs for this execution.

We want to determine such inputs by reasoning backwards from an abstract error state. The first step will be to define and implement a backward interpreter for an interesting subset of the TouchDevelop language features and API: Starting from the abstract error

state, we compute the backwards fixed point semantics, i.e. we basically execute the program backwards. This should narrow down the possible input states leading to the possible error. [7] suggests a similar approach, but they apply it to a completely different language for embedded software which for example does not make use of a heap. The key challenge here is to make the analysis precise enough, while keeping the approach sound. The soundness of some important abstract transformers will be proven.

Once the restrictions on the input states are computed by the backward analysis, there are three possible cases for the set of traces starting from these inputs:

1. The set is empty. Therefore we know that the reported error was a *false alarm* because the error state is not reachable.
2. All the traces in the set lead to the error state. We can pick any of them as a *counter-example*.
3. Some traces may reach the predicted state with an error, but not all. It is *unclear* which of them are counter-examples, if any. (Our approximated traces are not a subset of the error traces).

It is unclear how often each case will occur. We will perform a case study with a set of synthetic examples and all the currently available TouchDevelop scripts to evaluate the number of false alarms found as well as the number of counter-examples inferred.

Example

```
1  var w := wall->ask number("width?")
2  var h := wall->ask number("height?")
3  var radial := wall->ask boolean ("radial shape?")
4  var pic := media->create picture(w, h)
5  pic->post to wall
6  for 0 <= i1 < 50 do
7    if radial then
8      pic->draw text(w/2, h/2,
9             text, font, 0, colors -> rand)
10   else
11     pic->draw text(math ->rand(w), math->rand(w),
12            text, font, math -> rand(360), colors -> rand)
13 pic->update on wall
```

Consider the slightly modified code snipped from a published TouchDevelop script above. It lets the user create some kind of artistic image by drawing text in random colors and positions. The user enters the desired width, height and chooses among two drawing variants.

There are two problems a static analyzer may report here: Because the dimensions are not validated, negative values for the picture size can violate the precondition of `create picture` on line 4. A more subtle bug is present on line 11: Both the x and y-coordinates of the drawn text are chosen at random from $[0, w]$. In case $w > h$, text may be placed

vertically outside the picture bounds since $[0, w] \not\subseteq [0, h]$. Drawing outside bounds is often not desirable and an analysis could detect it in this case.

Ideally, our backward analysis should then produce a counter-example, e.g. for the second bug we may find the inputs $w = 400$, $h = 200$, $radial = false$ and the random coordinates happen to be $x = 300$, $y = 600$ (since $y > h$, text is out-of-bound).

The deliverables of the core include:

- Abstract semantics with (partial) soundness proof
- Concretization of input state to find counter examples
- Implementation
- Evaluation

Possible Extensions

Depending on the time left and progress of the project, there are several possible extensions:

- *Trace partitioning:* Normally static analyses perform a reachable states abstraction, therefore losing information about how a concrete program trace can arrive at a state. For example, in a state after an `if`-statement, it may be unclear which parts of it are due to the different branches taken. Trace partitioning techniques perform an abstraction over a partition of the set of traces to regain some precision lost when approximating the trace semantics [5].
- *Iterative refining approach:* To improve the precision of our analyzer, we can iteratively switch between forward and backward analysis. Different iteration strategies provide a lot of opportunity for fine-tuning. For example, if our backward analysis concludes that an `if`-branch is never executed, we can apply forward analysis again to get rid of the possibly imprecise join after the `if`.
- *Backwards semantics for complete API:* The TouchDevelop API is increasingly providing more and more features. We first focus on supporting the basic language features and parts of the API. For the rest, coarse but sound abstractions can be used. If time permits, it would be nice to implement precise abstract transformers covering the complete TouchDevelop API.

Project Management

Project Plan

The project is roughly divided into four phases:

- *Getting Started (first half of October):* Getting familiar with the environment, setting up all required tools, reading relevant papers.
- *Design and implementation of core (late October - January):* The backwards analysis will be designed, implemented and evaluated. In late December or January the intermediate presentation will be given. The deliverables include source code and a set of examples with expected results.
- *Work on extensions (January-February):* Depending on how much time is left, implement one or more extensions.
- *Writing (March):* Documenting the project work. This includes a final report and presentation.

Meetings

Weekly meetings will be held when possible, to discuss progress and define further steps to be taken.

References

[1] Giulia Costantini, Pietro Ferrara, and Agostino Cortesi. Static analysis of string values. In *Formal Methods and Software Engineering*, pages 505--521. Springer, 2011.

[2] Patrick Cousot and Radhia Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 238--252. ACM, 1977.

[3] Patrick Cousot and Radhia Cousot. Systematic design of program analysis frameworks. In *Proceedings of the 6th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 269--282. ACM, 1979.

[4] Pietro Ferrara, Raphael Fuchs, and Uri Juhasz. Tval+: Tvla and value analyses together. In *Software Engineering and Formal Methods*, pages 63--77. Springer, 2012.

[5] Laurent Mauborgne and Xavier Rival. Trace partitioning in abstract interpretation based static analyzers. In *Programming Languages and Systems*, pages 5--20. Springer, 2005.

[6] Microsoft. Touch Develop environment. `http://www.touchdevelop.com/`. [Online; accessed 29-September-2013].

[7] Xavier Rival. Understanding the origin of alarms in astrée. In *Static Analysis*, pages 303--319. Springer, 2005.

[8] Nikolai Tillmann, Michal Moskal, Jonathan de Halleux, and Manuel Fahndrich. Touchdevelop: Programming cloud-connected mobile devices via touchscreen. In *Proceedings of the 10th SIGPLAN symposium on New ideas, new paradigms, and reflections on programming and software*, pages 49--60. ACM, 2011.

[9] Matteo Zanioli, Pietro Ferrara, and Agostino Cortesi. Sails: static analysis of information leakage with sample. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, pages 1308--1313. ACM, 2012.