Invariant Generation for Complexity Analysis of Python Programs

Adam Blank (adamblan@cs) and Michael Arntzenius (daekharel@gmail.com)

Project Description

The high level main goals of this project are to implement several programs:

- `ub_time_complexity(P)` returns a valid symbolic upper bound on the *run time* of P.
- `ub_output_complexity(P)` returns a valid symbolic upper bound on the *size* of the output of P.
- `lb_time_complexity(P)` returns a valid symbolic lower bound on the *run time* of P.
- `lb_output_complexity(P)` returns a valid symbolic lower bound on the *size* of the output of P.

Just as a quick example, consider the following code:

```
def f(n):
    out = 1
    for i in range(n):
        out = out * n
    return out
```

We can see that `ub_timecomplexity(f)` $= n \in \mathcal{O}(n)$, because the loop runs n iterations; however, the function *returns* `out` which started as 1 and was multiplied by n, n times. So, `ub_outputcomplexity(f)` $= n^n \in \mathcal{O}(n^n)$.

0.1 Restrictions

In order to keep the scope of this assignment reasonable, we will restrict input programs in the following ways:

- All functions will be of type `int` $\to$ `int`.
- We will only simplify recursive programs of several particular forms.
- We will ignore certain dynamic features of Python.
- If termination/invariant detection is too difficult to infer, we will output `?`.

The difficulty arises in how to handle loops and recursion.

0.2 Loops

To effectively bound the runtime of a loop, we need to be able to bound the number of iterations. We propose doing this by using a variety of AI techniques and ideas from CITE PAPER HERE to generate invariants on the outputs of variables after multiple iterations. In particular, we aim to be able to handle *binary search*-like functions, and *early terminations* of loops. We intend to use search techniques on the AST of the loop, planning to prove the invariants we find correct, and possibly machine learning to classify programs as terminating or not.

0.3 Recursion

To effectively bound the runtime of a recursive function, we need to be able to bound the number of recursive calls. We only consider the class of programs where one of the following is true, for $f(\mathbf{x})$. We say $f(a_1, a_2, \ldots, a_n) \rightsquigarrow f(b_1, b_2, \ldots, b_n)$, when $f(\mathbf{b})$ is called as part of executing $f(\mathbf{a})$:

- $\exists i.\ \forall \mathbf{y}.\ f(\mathbf{x}) \rightsquigarrow f(\mathbf{y}) \implies y_i < x_i$
- $\exists (M \in \mathbb{N}).\ \forall f(\mathbf{y}).\ M \geq f(\mathbf{y})$ and $\exists i.\ \forall \mathbf{y}.\ f(\mathbf{x}) \rightsquigarrow f(\mathbf{y}) \implies y_i > x_i$

Again, to determine of the programs are of the form, and to prove such a fact, we use planning and search.

References