Homework 7

Jiří Klepl

We want to generalize the idea of checking of program equivalence to the situation where we can write to variables as well. First, I will give a general idea and then I will explain how to achieve it.

So far, we naively assumed the world is something which does never change. If we want to allow writes, this naive idea instantly falls apart.

What I propose is that we introduce a 'world' variable. Statements and procedure calls will take this world variable and output it as well. Function calls will take the world variable on top of their arguments and output a 2-value tuple consisting of its return value and the world variable. All other language construct will follow this basic idea. (We don't have to follow this idea literally, just semantically)

Every write to a certain address will change the world variable and every read from an address will retrieve a value from the world variable. Previously, we would translate a program:

```
a->data = 1;
x = a->data;
a->data = 2;
x = a->data;
```

as: $data(a) = 1 \wedge x = data(a) \wedge data(a) = 2 \wedge x_1 = data(a)$ (notice the quasi-SSA form)

I propose a use of two functions: $W(addr, val, world) \to world'$ and $R(addr, world) \to val$ for writes and reads, respectively (and T for tuples), and W_i for the world variables; and thus rewrite the program as follows (simplified as we know the *data* accessor has no side-effects):

$$W_{i+1} = W(data(R(a, W_i)), 1, W_i) \wedge$$
$$W_{i+2} = W(x, R(data(R(a, W_{i+1})), W_{i+1}), W_{i+1}) \wedge$$
$$W_{i+3} = W(data(R(a, W_{i+2})), 2, W_{i+2}) \wedge$$
$$W_{i+4} = W(x, R(data(R(a, W_{i+3})), W_{i+3}), W_{i+3})$$

This idea works for **if** statements too and we can use recursion, this gives us an ability to represent an arbitrary algorithm. All we need is a good set of additional axioms:

- independent writes: $a \neq b \to W(a, x, W(b, y, W_i)) = W(b, y, W(a, x, W_i))$
- overwrite: $W(a, x, W(a, y, W_i)) = W(a, x, W_i)$
- write and read: $R(a, W(a, x, W_i)) = x$
- unnecessary write : $R(a, W_i) = x \to W(a, x, W_i) = W_i$ (prevents hidden side-effects)
- (*bonus, if we want*) safe memory: $R(a, W_0) = NULL$

Then, all we need to specify is that local variables have distinct addresses. For heap, we can use a malloc function which returns a fresh address from each call. This is easy if we associate local variable addresses with positive values and heap-allocated addresses with negative, for example. For auto storage variables, we can use a stack counter.