Incorrectness Specification Inference

ANONYMOUS AUTHOR(S)

1 MOTIVATION EXAMPLE: CONCAT

For the concat example,

```
let rec concat s1 s2 =
  match s1 with
  | [] -> s2
  | h1 :: t1 ->
    let s3 = concat t1 s2 in
    h1 :: s3
```

we expect that the overapproximate triple $\{\Sigma\} concat\ s_1\ s_2 \downarrow v \{\Phi\}$ hold, where:

$$\Sigma \equiv \neg dup(s_1) \land \neg dup(s_2)$$
$$\Phi \equiv \neg dup(v)$$
$$dup(s) \equiv \text{ the stack } s \text{ contains some elements appearing for 2 or more times.}$$
$$emp(s) \equiv \text{ the stack } s \text{ is empty.}$$

However, this triple is not valid (e.g. $concat\ [1; 2]\ [2; 3] = [1; 2; 2; 3]$) which means the program is buggy. Thus we expect to find an underapproximate triple as its incorrectness specification that summarize the buggy executions of $concat$. We cannot simply negate the postcondition Φ to build a triple $[\Sigma] concat\ s_1\ s_2 \downarrow v [\neg\Phi]$ because there exists unreachable state in the negated postcondition $\neg\Phi$. For example, the stack $v \equiv [1; 1; 1]$ is not reachable, because all elements in the output stack should be contained by the two input stacks s_1 and s_2, additionally there are three 1s in the output stack, thus at least one of the input stack will contains two 1s which conflicts with the precondition Σ that asks both input stacks having no duplicate element.

1.1 Problem

Thus our goal is to infer P and Q such that:

(1) $[P] concat\ s_1\ s_2 \downarrow v [Q]$;
(2) $P \implies \Sigma$;
(3) $Q \implies \neg\Phi$.

1.2 Solution

One possible result is:

$$P_0 \equiv \neg dup(s_1) \land \neg dup(s_2) \land \exists u, mem(s_1, u) \land mem(s_2, u)$$
$$Q_0 \equiv dup(v) \land \neg dup3(v)$$

where $dup3(s)$ means the stack s contains some elements appearing for 3 or more times.

2022. XXXX-XXXX/2022/5-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1.3 Proof fails without inductive invariant

Although P_0 and Q_0 is correct but we cannot verify the triple $[P_0]concat\ s_1\ s_2 \downarrow v[Q_0]$ directly, because P_0 and Q_0 is not inductive. More precisely, we expect that,

$$[P]T[ok : Q]$$

where T is an imperative version (but with recursion) of $concat$:

$$\begin{aligned}
&T := \\
&\quad (\text{assume } s_1 = []; v := s_2)\oplus \\
&\quad (\text{assume } \neg(s_1 := []); (h_1, t_1) := cons^{-1}(s_1); s_3 := T(t_1, s_2); v := h_1 :: s_3)
\end{aligned}$$

Now we do not consider exceptions, thus we label Q as ok (and omitted if the context is clear) and expect the program can terminate normally but reach state over from Φ. Now we show, if we do not introduce the new inductive invariant, the proof will fail.

Notice that, the proof tree actually starts from the post condition $P(s_1, s_2) \wedge Q(v)$ instead of $Q(v)$, because the incorrectness logic need constraint all variables in the postcondition. Thus the input arguments should be constrained by the precondition $P(s_1, s_2)$. In summary, we add a new rule:

$$\dfrac{[P(\overline{x})]\overline{y} = T(\overline{x})[P(\overline{x}) \wedge Q(\overline{y})]}{Spec(T, (P, Q))}\ \text{Specification}$$

Daully, we add an application rule:

$$\dfrac{Spec(T, (P, Q))}{\forall \overline{x}\ \overline{y}, [P(\overline{x})]\overline{y} = T(\overline{x})[P(\overline{x}) \wedge Q(\overline{y})]}\ \text{Apply}$$

Now we have

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\begin{array}{l}[P(s_1, s_2)] \\ \text{assume } \neg(s_1 := []); \\ [P(s_1, s_2) \wedge s_1 \neq []]\end{array}}\ \text{Assume} \qquad \dfrac{\dfrac{}{\begin{array}{l}[P(s_1, s_2) \wedge s_1 \neq []] \\ (h_1, t_1) := cons^{-1}(s_1) \\ [P(s_1, s_2) \wedge s_1 \neq [] \wedge s_1 = h_1 :: t_1]\end{array}}\ \text{Assign} \qquad \begin{array}{l}[P(s_1, s_2) \wedge s_1 \neq [] \wedge s_1 = h_1 :: t_1] \\ s_3 := T(t_1, s_2); v := h_1 :: s_3 \\ [\exists ... \wedge Q(v)]\end{array}}{[P(s_1, s_2) \wedge s_1 \neq []](h_1, t_1) := cons^{-1}(s_1); ...[\exists ... \wedge Q(v)]}\ \text{Seq}}{[P(s_1, s_2)]\text{assume } \neg(s_1 := []); (h_1, t_1) := cons^{-1}(s_1); ...[\exists ... \wedge Q(v)]}\ \text{Seq}}{\dfrac{[P(s_1, s_2)]T(s_1, s_2) = v[\exists s_3\ h_1\ t_1, P(s_1, s_2) \wedge Q(v)]}{Spec(T, (P, Q))}\ \text{Spec}}\ \text{Choice: 1}$$

We choose the second branch of T, because the first branch will lead $P(s_1, s_2) \wedge s1 = [] \iff \perp$.

$$\dfrac{\dfrac{\dfrac{Spec(T, (P, Q))}{\begin{array}{l}[P(t_1, s_2)]s_3 := \\ T(t_1, s_2)[Q(s_3)]\end{array}}\ \text{Apply} \qquad \color{red}{\begin{array}{l}P(t_1, s_2) \implies \\ P(s_1, s_2) \wedge s_1 \neq [] \wedge s_1 = h_1 :: t_1\end{array}}}{\begin{array}{l}[P(s_1, s_2) \wedge s_1 \neq [] \wedge s_1 = h_1 :: t_1]s_3 := \\ T(t_1, s_2) \\ [P(s_1, s_2) \wedge s_1 \neq [] \wedge s_1 = h_1 :: t_1 \wedge Q(s_3)]\end{array}}\ \text{Cons} \qquad \dfrac{\dfrac{}{\begin{array}{l}[... \wedge Q(s_3)] \\ v := h_1 :: s_3 \\ [.. \wedge Q(s_3) \wedge v = h_1 :: s_3]\end{array}}\ \text{Assign} \qquad \color{red}{\begin{array}{l}\exists s_3\ h_1\ t_1, P(s_1, s_2) \wedge Q(v) \implies \\ P(s_1, s_2) \wedge s_1 \neq [] \wedge s_1 = h_1 :: t_1 \\ \wedge Q(s_3) \wedge v = h_1 :: s_3\end{array}}}{\begin{array}{l}[P(s_1, s_2) \wedge s_1 \neq [] \wedge s_1 = h_1 :: \\ t_1 \wedge Q(s_3)]\ v := h_1 :: s_3 \\ [\exists ... \wedge Q(v)]\end{array}}\ \text{Con}}{\begin{array}{l}[P(s_1, s_2) \wedge s_1 \neq [] \wedge s_1 = h_1 :: t_1] \\ s_3 := T(t_1, s_2); v := h_1 :: s_3\ [\exists ... \wedge Q(v)]\end{array}}\ \text{Seq}$$

When we encounter the recursive calling $T(t_1, s_2)$, to apply the triple $[P(t_1, s_2)]s_3 := T(t_1, s_2)[Q(s_3)]$ to the current state, we use the consequence rule of the incorrectness logic. It requires

$$P(t_1, s_2) \implies P(s_1, s_2) \wedge s_1 \neq [] \wedge s_1 = h_1 :: t_1$$

However, we cannot prove it, as we do not know if $\exists u, mem(s_1, u) \wedge mem(s_2, u)$ (required by $P(s_1, s_2)$). One counter example is

$$\begin{aligned}
&s_1 \equiv [1; 1], s_2 \equiv [1; 3], \\
&h_1 \equiv 1, t_1 \equiv [1]
\end{aligned}$$

which means the precondition $P(s_1, s_2)$ should be strengthen (or $P(s_1, s_2)$ should be weaken). On the other hand, we encounter another similar implication when we try to finish the proof:

$$P(s_1, s_2) \wedge Q(v) \implies P(s_1, s_2) \wedge s_1 \neq [] \wedge s_1 = h_1 :: t_1 \wedge Q(s_3) \wedge v = h_1 :: s_3$$

Still, we cannot prove it, as we do not know if $dup(s_3)$ (required by $Q(s_3)$). One counter example is:

$$s_1 \equiv [1; 2], s_2 \equiv [1; 3], v = [1; 2; 1; 3]$$
$$h_1 \equiv 1, t_1 \equiv [2], s_3 = [2; 1; 3]$$

which means the postcondition $Q(v)$ should be strengthen (or $Q(s_3)$ should be weaken). Let's look deeper to make the problem clear. This counterexample will happen during the execution (unreachable from *concat* $[1; 2]$ $[1; 3]$). Thus we expect $Q(s_3)$ to be consistent with $[2; 1; 3]$:

$$s_3 \equiv [2; 1; 3] \models Q(s_3) \tag{1}$$

However, we also want $Q(s_3) \implies \neg\Phi$, thus

$$s_3 \equiv [2; 1; 3] \models dup(s_3) \tag{2}$$

which is impossible, thus (P, Q) is not inductive,

1.4 Cannot find an inductive invariant

Assume there exists an inductive invariant (P_I, Q_I) can help to prove (P, Q). Following the conseqeunce rule, we expect:

$$\frac{P_I \implies P \qquad [P_I]C[Q_I] \qquad Q \implies Q_I}{[P]C[Q]}\ \text{Con}$$

where

$$P_I(s_1, s_2) \wedge Q_I(v) \implies P_I(s_1, s_2) \wedge s_1 \neq [] \wedge s_1 = h_1 :: t_1 \wedge Q_I(s_3) \wedge v = h_1 :: s_3$$

Notice that, the new postcondition Q_I is not required to be a subset of $\neg\Phi$. There are two reachable paths:

$$s_1 \equiv [1; 2], s_2 \equiv [1; 3], v = [1; 2; 1; 3]$$
$$h_1 \equiv 1, t_1 \equiv [2], s_3 = [2; 1; 3]$$

and

$$s_1 \equiv [2; 1], s_2 \equiv [1; 3], v = [2; 1; 1; 3]$$
$$h_1 \equiv 2, t_1 \equiv [1], s_3 = [1; 1; 3]$$

Thus we expect Q_I consistent with these four concrete values: $[2; 1; 3]$, $[1; 2; 1; 3]$, $[1; 1; 3]$ and $[2; 1; 1; 3]$. Then Q_I contains both duplicate stacks and non-duplicate stacks, and if a stack s_3 can satisfy Q_I dependents on if h_1 is a member of s_3. If we do not know the value of h_1, we cannot decide if a given stack can satisfy Q_I. However, h_1 is not belong to the arguments or the result of calling $s_3 = T(t_1, s_2)$.

ACKNOWLEDGEMENTS

REFERENCES