Separation Logic in Infer

Operational Semantics

How to establish memory model in the presence of pointers?

- By Hoare Triple

$$\{\mathsf{emp}\}\mathbf{malloc}()\{(\mathsf{ret}\mapsto -) \vee (\mathsf{ret} = \mathsf{nil} \wedge \mathsf{emp})\} \qquad \{x\mapsto -\}\mathbf{free}(x)\{\mathsf{emp}\}$$
$$\{x\mapsto - \wedge y = Y\}[x] := y\{x\mapsto Y \wedge y = Y\} \qquad \{x\mapsto X\}\mathtt{return}[x]\{x\mapsto X \wedge \mathsf{ret} = X\}$$

Basic Idea

- It analyzes programs by generating and composing function specifications in a bottom-up manner by means of bi-abduction. The specification of a function is a set of Hoare Tripes {{P} func() {Q}}, where P is the weakest pre-condition of the safe execution of the function *func*. A function and its callees are bug-free if its specification set is not empty, indicating that these functions are executed without bugs under some preconditions.
- How to achieve context sensitive?
    - The context-sensitivity is achieved by *inferPre* and *inferSpec* illustrated by Algorithm 4 and 5 in his paper of short version(POPL 2009). The inferred specifications of the callees are inlined to get the weakest precondition and postcondition of the caller.
- How to achieve path sensitive?
    - This work does not support path sensitivity thoroughly. If the branch condition occurs in the weakest precondition at the entry of the branch after the condition, the effect of the branch condition is considered by removing the conjunction from the formula to generate the better weakest precondition, like the following first example shows. Otherwise, the branch condition will be skipped, just as the second example shows. Because the precondition is not weakest anymore, this trick can cause false alarms when the failure in generating specifications of its caller arises.

```
void why_assume_as_assert(struct node *x)       {
    if (x!=0)   { x->tl = 42;  }      }
```

emp

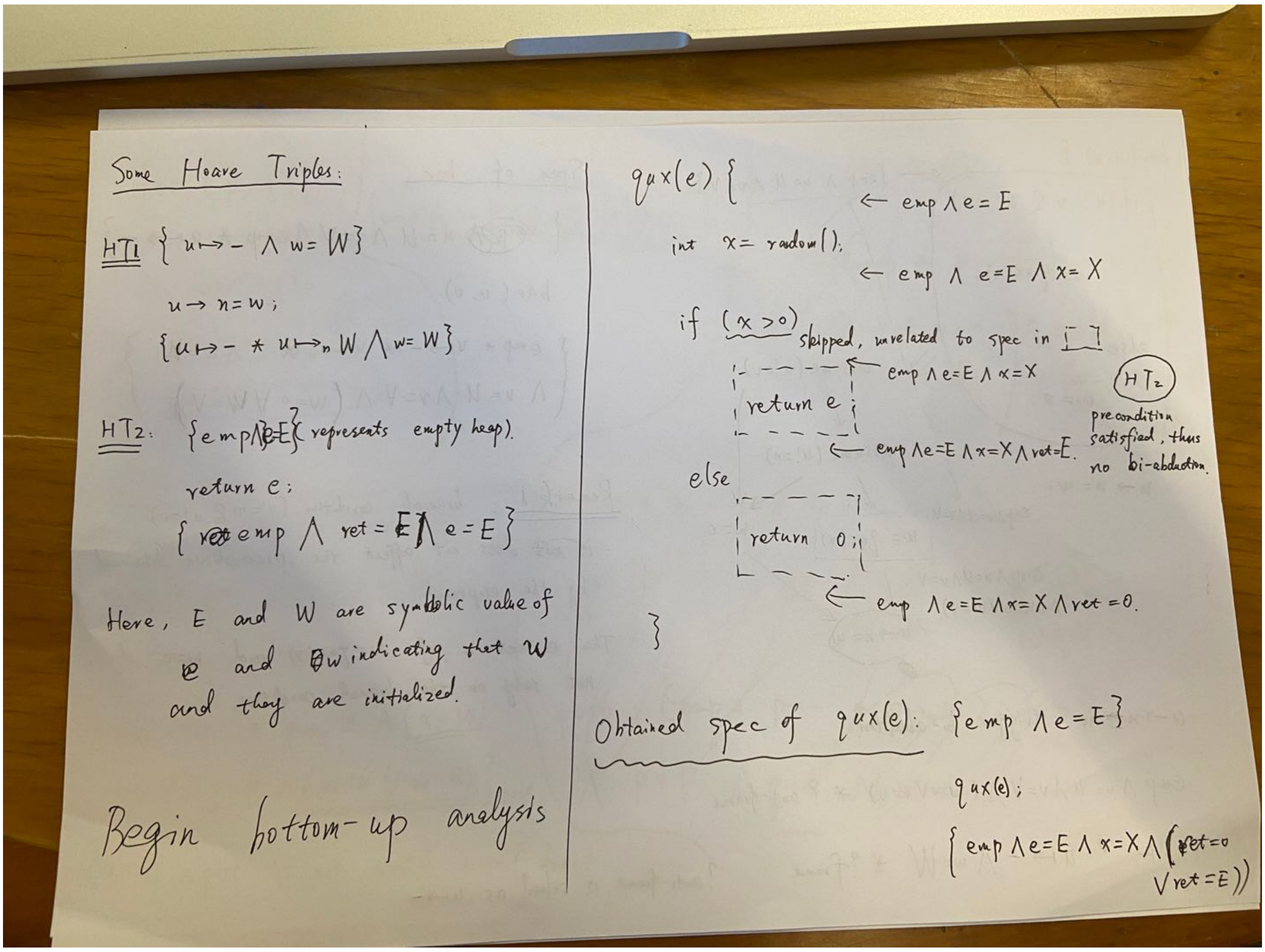
x!=0

```
void nondet-example(struct node *x,*y) {
    if (nondet()) {x->tl = 0;}
```

x!=0

Difficult to model

An Example



Some Hoare Triples:

HT1 $\{ u \mapsto - \wedge w = W \}$

$u \to n = w;$

$\{ u \mapsto - * u \mapsto_n W \wedge w = W \}$

HT2: $\{ emp \wedge e = E \}$ (emp represents empty heap).

return e;

$\{ emp \wedge ret = E \wedge e = E \}$

Here, E and W are symbolic value of e and w indicating that w and they are initialized.

Begin bottom-up analysis

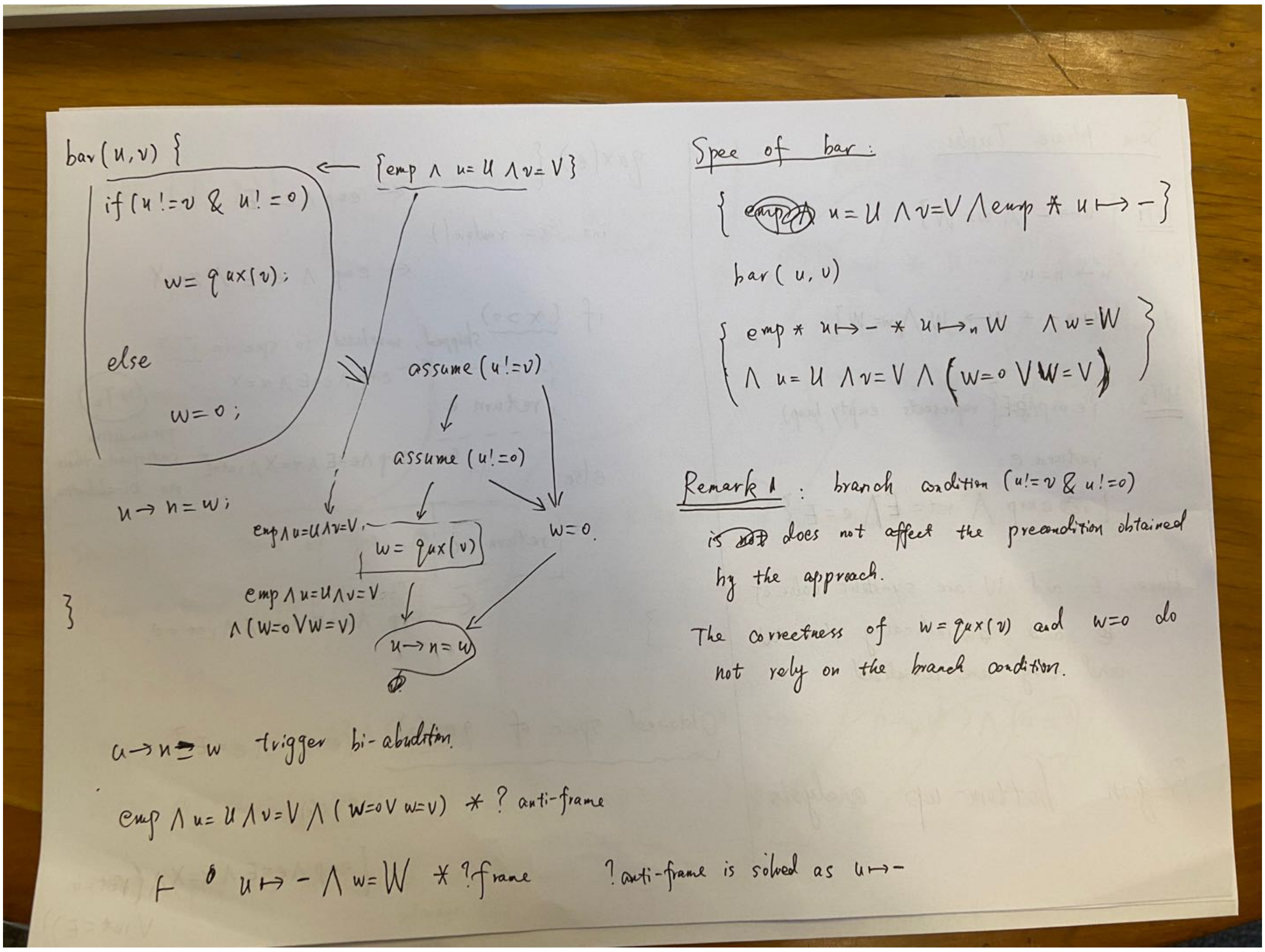
```
qux(e) {                 ← emp ∧ e = E
  int x = random();
                         ← emp ∧ e = E ∧ x = X
  if (x > 0)             skipped, unrelated to spec in [ ]
                         ← emp ∧ e = E ∧ x = X
     return e;
                         ← emp ∧ e = E ∧ x = X ∧ ret = E.
  else
     return 0;
                         ← emp ∧ e = E ∧ x = X ∧ ret = 0.
}
```

HT2 precondition satisfied, thus no bi-abduction.

Obtained spec of qux(e): $\{ emp \wedge e = E \}$

qux(e);

$\{ emp \wedge e = E \wedge x = X \wedge (ret = 0 \vee ret = E) \}$



```
bar(u,v) {               ← {emp ∧ u = U ∧ v = V}
  if (u != v & u != 0)
     w = qux(v);
  else
     w = 0;
  u→n = w;
}
```

assume (u!=v)

assume (u!=0)

$emp \wedge u = U \wedge v = V$

$\boxed{w = qux(v)}$

w = 0.

$emp \wedge u = U \wedge v = V \wedge (w = 0 \vee w = V)$

$u \to n = w$

$u \to n = w$ trigger bi-abduction.

$emp \wedge u = U \wedge v = V \wedge (w = 0 \vee w = v) * ?\text{anti-frame}$

$\vdash u \mapsto - \wedge w = W * ?\text{frame}$?anti-frame is solved as $u \mapsto -$

Spec of bar:

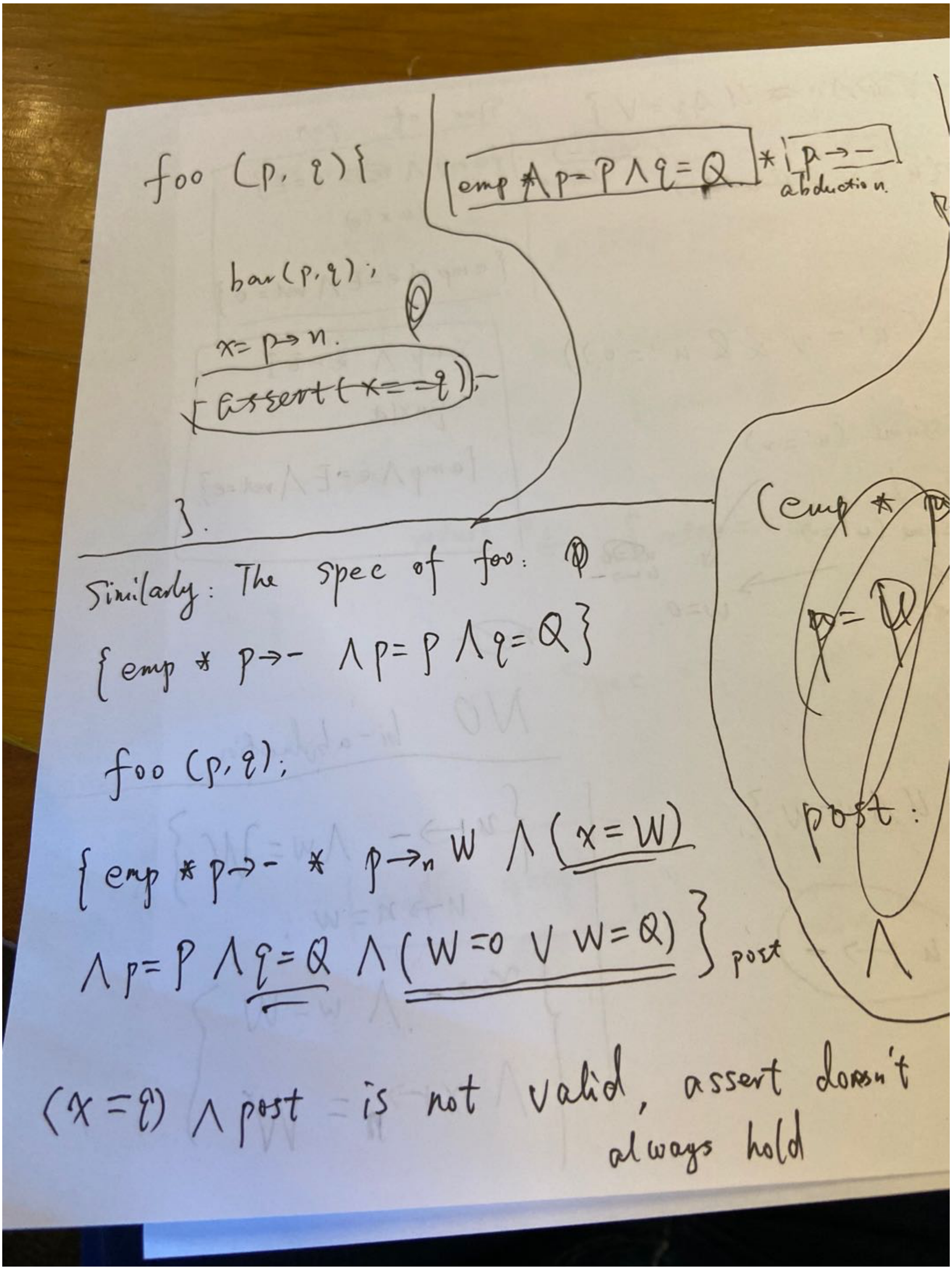
$\{ u = U \wedge v = V \wedge emp * u \mapsto - \}$

bar(u, v)

$\{ emp * u \mapsto - * u \mapsto_n W \wedge w = W \wedge u = U \wedge v = V \wedge (w = 0 \vee W = V) \}$

Remark 1: branch condition (u!=v & u!=0) does not affect the precondition obtained by the approach.

The correctness of w = qux(v) and w = 0 do not rely on the branch condition.



In this example, the specifications of functions are generated in a bottom-up manner, where the specification of the function is a collection of Hoare Triples `{{P} fun() {Q}}`. Specifically, P and Q are the weakest pre-condition and post-condition of `func` in safe executions. Meanwhile, The weakest precondition and postcondition of each statement can be obtained in this process, and they can be used to verify or invalidate the assertion. For example, the post-condition in the third picture shows that the `assert(x==q)` does not hold for all the executions. This is consistent to the actural execution of the program.

However, the analysis degrades the precision of the specification of `bar` . The specification obtained in the first picture shows that the safe execution of `qux` does not rely on the branch condition. The heuristic trick handling paths of their approach ignore the effect of the branch conditions, so the post-condition of `bar` presumes the cases that `U=V=W`, which do not occur in the actural run.

Another issue is manual effort for terminality and avoiding explosive size of separation logic formulae. The statements in the forms of `x->n = y` and `x=y->n` increase the size of a logic formula in the presence of long and even unbounded program paths, such as loops and recursive functions. Specific folding rules are needed to eliminate variables in the formula. For example, `ls(x, y) * ls(y, z) * ls(z, nil)` is replaced by `ls(x, nil)` if `ls(x, nil) = (x = nil) \lor (\exists y, ls(x, y) * ls(y, nil))` is defined in the analysis of list manipulating programs. The folding rules differs for different data structures, and people need to write these rules for their programs.

Last by not least, except for the fragment of linked list [3], decision problem of general separation logic is not decidable [2]. The limitation of solvers makes the bi-aduction based approaches not strong as it would be, because some heuristics are introduced to assure that the analysis can terminate under its time setting.

Reference

[1] Calcagno C, Distefano D, O'hearn P W, et al. Compositional shape analysis by means of bi-abduction[J]. Journal of the ACM (JACM), 2011, 58(6): 1-66.

[2] Berdine J, Calcagno C, O'hearn P W. Symbolic execution with separation logic[C]//Asian Symposium on Programming Languages and Systems. Springer, Berlin, Heidelberg, 2005: 52-68.

[3] J. Berdine, C. Calcagno, and P. W. O'Hearn. A decidable fragment of separation logic. In FSTTCS 2004, volume 3328 of LNCS, pages 97–109. Springer, Dec. 2004.