March 10, 2016

Algorithm 1: Recursive repair generation algorithm with Program Sketch

```
Input   : Assertion set $a\_set$, Function Set $f\_set$, A set of transformation functions $s\_set$
Output: Updated $\langle a\_set, f\_set \rangle$ with maximum number of assertions pass
$e\_set = \phi$; //expression set
$l\_set = \phi$; //location set
$f\_set' = \phi$; //newly generated functions
$a\_set' = \phi$; //selected assertions
$r\_set = \phi$; //repair set
$a\_fail = null$; //assertion failure
Function  recurRepair ($\langle a\_set, f\_set \rangle$) : $\langle a\_set, f\_set \rangle$ is
    $failure$ = runSketch($\langle a\_set, f\_set \rangle$);
    if $failure == a\_fail$ then
        return $\langle a\_set, f\_set \rangle$; //stop recursion
    end
    $a\_fail = failure$;
    $field$ = findSuspicious($a\_fail$);
    $type$ = typeOf($field$);
    $a\_set'$ = locateInHarness($type$);
    $f\_set' = \phi$;
    foreach schema $s \in s\_set$ do
        foreach function $f \in f\_set$ do
            $l\_set$ = locateInFunction($f, field$);
            if $l\_set == null$ then
                $f\_set' = f\_set' \cup f$;
            else
                $e\_set$ = findExpInFunction($f, type$);
                $r\_set = \phi$;
                foreach location $l \in l\_set$ do
                    $\langle l, f\_add \rangle = createUpdate(s, l, e\_set)$;
                    $r\_set = r\_set \cup \langle l, f\_add \rangle$;
                end
                $f'$ = replaceFunc($f, r\_set$);
                $f\_set' = f\_set' \cup f'$;
            end
        end
        return recurRepair($\langle a\_set', f\_set' \rangle$);
    end
end
```

Algorithm 2: Repair template generation algorithm

```
Function createUpdate ($s, l, e\_set$) : $\langle l, f\_add \rangle$ is
    //define four schema here.
end
def s1($l, e\_set$) //rhs only:
    $rhs$ = concat($e\_set$) ;
    if typeOf(e_set is primitive type) then
        $rhs = rhs$+??;
    else
        $rhs = rhs + null$;
    end
    return $lhs(l) = rhs$;
end
def s1($l, e\_set$) //both lhs and rhs:
    $lhs$ = concat($e\_set$) ;
    if typeOf(e_set is primitive type) then
        $rhs = lhs$+??;
    else
        $rhs = lhs + null$;
    end
    return $lhs = rhs$;
end
def s1($l, e\_set$) // add one condition:
    $a$ = concat($e\_set$) ;
    if typeOf(e_set is primitive type) then
        $b = a$+??;
    else
        $b = a + null$;
    end
    $cond = concat('a == b', 'a! = b', 'true')$;
    return concat ($cond, a = b$);
end
```

Algorithm 3: Repair expression generation algorithm

```
Function findExpInFunction ($f, type$) : $e\_set$ is
    $e\_set = \phi$; //expression set
    $other\_queue = \phi$; //other type structs;
    foreach location $l \in l\_set$ do
        if defineTypeAt($l, type$) then
            $e\_set = e\_set \cup$ record($l$);
        else
            $other\_queue = other\_queue \cup$ record($l$);
        end
    end
    while other_queue not Empty do
        r = $poll(other\_queue)$;
        foreach field $fd \in r$ do
            if typeOf($fd$) == type then
                $e\_set = e\_set \cup field$;
            end
        end
    end
end
```

Algorithm 4: Repair driver algorithm

```
Input   : Sketch file $file$, A set of transformation functions $s\_set$
Output: Repaired SketchFile $file'$
Function recurRepair ($\langle a\_set, f\_set \rangle$) : $\langle a\_set, f\_set \rangle$ is
    $a\_set = \phi$; //assertion set
    $e\_set = \phi$; //expression set
    $f\_set = \phi$; //function set
    $l\_set = \phi$; //location set

    $a\_set$ = allHarness($file$);
    $f\_set$ = allFunctions($file$);
    $\langle a\_set', f\_set' \rangle = recurRepair(\langle a\_set, f\_set \rangle)$;
    $file'$ = createFile($\langle a\_set', f\_set' \rangle$);
end
```