《ESP: Path-Sensitive Program Verification in Polynomial Time 》Review

Paper Info

ESP: Path-Sensitive Program Verification in Polynomial Time Manuvir Das, Sorin Lerner, Mark Seigle

Major Contribution

The authors present a new algorithm for partial program verfication that can run in polynomial time and space. Previous work on partial verification has only focused on path-sensitive analysis methods, which limits the applicability to large programs.

The authors propose "property simulation" based on the heuristic that a branch is likely to be relevant only if the property FSM transitions to different states along the arms of the branch. This method avoids exponential blowup and only captures relevant branching behavior. This is the major contribution of this research paper, and the core insight and algorithm make this work state of art.

Main Work

The authors present property analysis, a general framework for tracking property states and execution states using path-sensitive dataflow analysis. The framework consists of two parts: intra-procedural property analysis and inter-procedural property analysis.

Intra-procedural property analysis

The intra-procedural property analysis in this paper is based on the traditional waiting-list algorithm in the dataflow analysis.

```
   global
1    Worklist : 2^N
2    Info : E → 2^S

   procedure Solve(CFG = [N, E])
   begin
3    for each e ∈ E do Info(e) := {}
4    Info(Out_T(n_entry)) := {[$uninit, ⊤]}
5    Worklist := {dst(out_T(n_entry))}

6    while Worklist ≠ ∅ do
7      Remove a node n from Worklist
8      switch(n)
9        case n ∈ Merge:
10         ss = F_mrg(n, Info(In_0(n)), Info(In_1(n)))
11         Add(Out_T(n), ss)
12       case n ∈ Branch:
13         ss_T = F_br(n, Info(In_0(n)), true)
14         ss_F = F_br(n, Info(In_0(n)), false)
15         Add(Out_T(n), ss_T)
16         Add(Out_F(n), ss_F)
17       case n ∈ Other:
18         ss = F_oth(n, Info(In_0(n)))
19         Add(Out_T(n), ss)

20   return Info
   end
   procedure Add(e, ss)
   begin
21   if Info(e) ≠ ss then
22     Info(e) := ss
23     Worklist := Worklist ∪ {dst(e)}
   end
```

The above algorithm is similar to the traditional data-flow framework. In this paper, the authors assume that each merge point only has two predecessors, and each branch point only has two successors. Hence, in the above algorithm, we only need to process two nodes when we encounter the merge points or branch points.

Based on the intra-procedural property analysis framework, a framework for property simulation can be easily implemented by chosing different domain of execution states and the join operations. In this paper, the authors propose three different kinds of frameworks, including the fully path-sensitive analysis, standard dataflow analysis and property simulation. Some examples are explained in the paper, and the details of these three frameworks are fully illustrated.

Inter-procedural property analysis

The inter-procedural property analysis is a extension of the intra-procedural version through the use of partial transfer functions or function summary edges. The algorithm framework is following.

```
   procedure Solve(Global CFG = [N, E, F])
   begin
4    for each [f, d] ∈ F × E do Summary(f, d) := ∅
5    for each [e, d] ∈ E × D do Info(e, d) := ∅
6    e := Out_T(entryNode(main))
7    Info(e, $uninit) := {[$uninit, ⊤]}
8    Worklist := {[e, $uninit]}

9    while Worklist ≠ ∅ do
10     Remove a pair [n, d] from Worklist
11     switch(n)
12       case n ∈ Call:
13         ss_in := Info(In_0(n), d)
14         ss_out := ∅
15         for each d' ∈ D s.t. ss_in[d'] ≠ ∅ do
16           if Summary(callee(n), d') ≠ ∅ then
17             ss_out := ss_out ∪ Summary(callee(n), d')
18           AddTrigger(entryNode(callee(n)), d', ss_in[d'])
19         Add(Out_T(n), d, α_as(ss_out))
20       case n ∈ Exit:
21         ss_in := Info(In_0(n), d)
22         AddToSummary(n, d, ss_in)
23       case n ∈ Merge:
24         ss_out := F_mrg(n, Info(In_0(n), d), Info(In_1(n), d))
25         Add(Out_T(n), d, ss_out)
26       case n ∈ Branch:
27         ss_T := F_br(n, Info(In_0(n), d), true)
28         ss_F := F_br(n, Info(In_0(n), d), false)
29         Add(Out_T(n), d, ss_T)
30         Add(Out_F(n), d, ss_F)
31       case n ∈ Other:
32         ss_out := F_oth(n, Info(In_0(n), d))
33         Add(Out_T(n), d, ss_out)

34   return Info
   end
```

As shown above, the inter-procedural property analysis merges the information in the function summary edges at the function call sites. The main part is similiar to the intra-procedural version.

In the evaluation section, the authors conduct some experiments on the gcc source code, and analyze the validity of the usage of some file operations. The results demonstrate that property simulation has a good performance both in the precision and efficiency.

Future Work

Some further exploration can be conducted in the future. Although the frameworks proposed in this paper has a good scalability, the parallel algorithm has not been desiged in this work. There are many phases which can be implemented in a parallel form, such as VFG construction and interface expression computation. Meanwhile, some infeasible paths can be filtered when we encounter a branch node. Some methods in the Pinpoint paper can be referred and applied to this work.