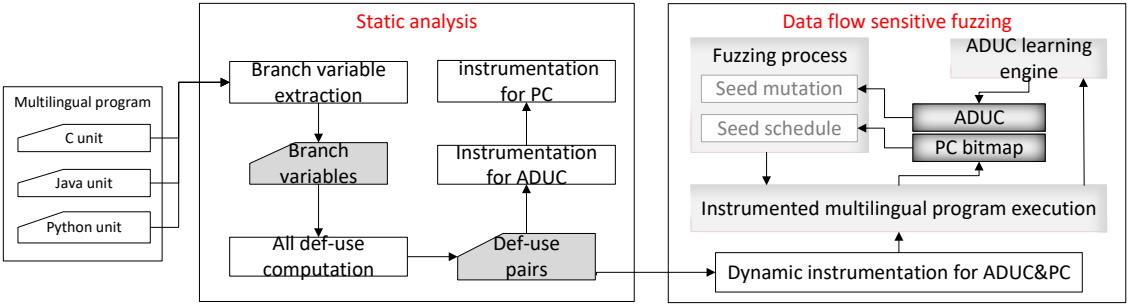
Overview



PC: path coverage, **ADUC**: all def-use coverage

Step1: (Control) cross-language coverage guided fuzzer based on AFL++.
Step2: (Data) Introduce data flow facts (Data adapter in runtime).
Step3: Fuzzing algorithm optimization (seed selection / mutation schedule)

Example

```
1 int Process (int List) {
2    int Len = len (List);
3    if (Len == 0) {
4                return 0;
5    }
6    int Res = 1;
7    for (int i = 0; i < Len; i++) {
8       int Val = List[i];
9       if (Val == 1) {
10               Res += Val;
11      }
12      else if (Val == 2) {
13               Res *= Val;
14      }
15      else
16      {
17               Res += Res/Val;
18      }
19   }
20   return Res;
21 }
```

Branch variables:

Len:
- Def (s2) -> P-Use (s3)

i:
- Def (s7) -> P-Use (s7)

Val:
- Def (s8) -> P-Use (s9)
- -> C-Use (s10)
- -> P-Use (s12)
- -> C-Use (s13)
- -> C-Use (s17)

List0 = [1,2,3,4,⋯,100]
-> Val = 100

Mutation by item:
List1 = [11,2,3,4,⋯,100]
-> Val = 100
List2 = [1,21,3,4,⋯,100]
-> Val = 100
……
List1023 = [1,2,3,4,⋯,991, 100]
-> Val = 100

List1023 = [1,2,3,4,⋯,99, 1001]
-> Val = 1001

=> Val is only determined by List[99]

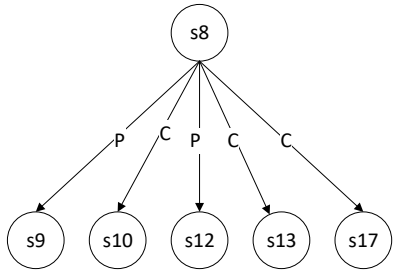
List0 = [1,2,3,4,⋯,100]
-> Val = ∑ Hash(List1[i])

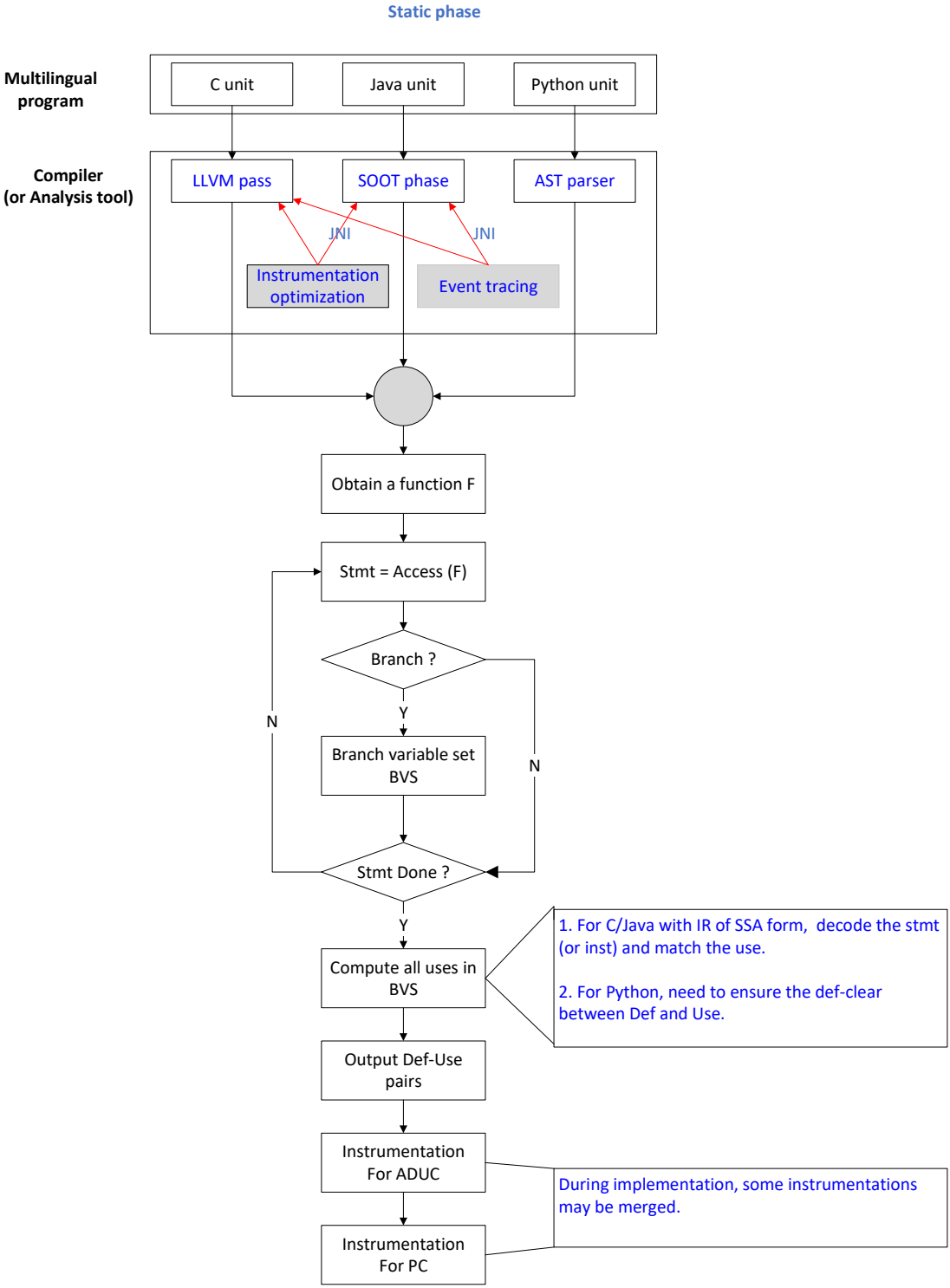
Mutation by item:
List1 = [11,2,3,4,⋯,100]
-> Val = ∑ Hash(List1[i])
List2 = [1,21,3,4,⋯,100]
-> Val = ∑ hash(List1[i])
……
List1023 = [1,2,3,4,⋯,991, 100]
-> Val = ∑ Hash(List1[i])

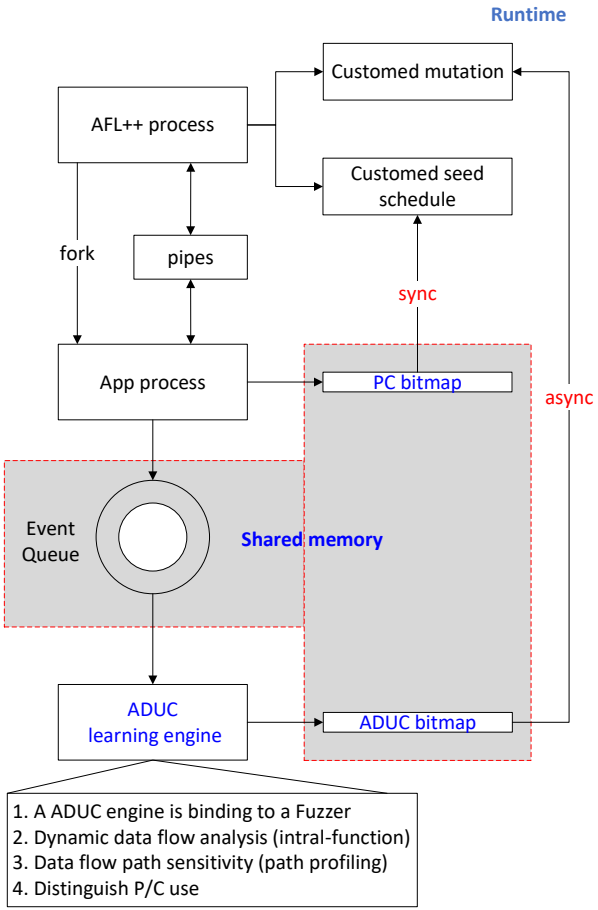
List1023 = [1,2,3,4,⋯,99, 1001]
-> Val = ∑ Hash(List1[i])

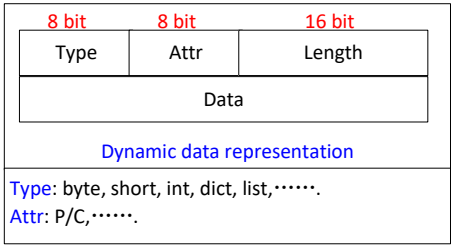
=> Val is determined by each List[i]
=> ADUC change -> seed schedule
=> P-Use -> seed mutation
=> C-Use -> directed fuzzing

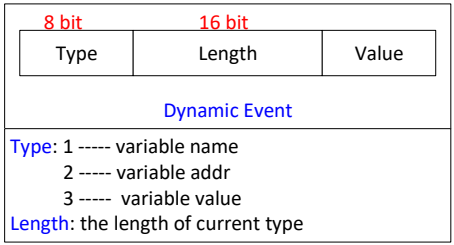
Value = hashed bitmap




Static phase
Multilingual program
C unit
Java unit
Python unit
Compiler (or Analysis tool)
LLVM pass
SOOT phase
AST parser
JNI
JNI
Instrumentation optimization
Event tracing
Obtain a function F
Stmt = Access (F)
Branch ?
Y
N
Branch variable set BVS
N
Stmt Done ?
Y
Compute all uses in BVS
1. For C/Java with IR of SSA form, decode the stmt (or inst) and match the use.
2. For Python, need to ensure the def-clear between Def and Use.
Output Def-Use pairs
Instrumentation For ADUC
Instrumentation For PC
During implementation, some instrumentations may be merged.


Runtime
Customed mutation
AFL++ process
Customed seed schedule
fork
pipes
sync
App process
PC bitmap
async
Event Queue
Shared memory
ADUC learning engine
ADUC bitmap
1. A ADUC engine is binding to a Fuzzer
2. Dynamic data flow analysis (intral-function)
3. Data flow path sensitivity (path profiling)
4. Distinguish P/C use


8 bit
8 bit
16 bit
Type
Attr
Length
Data
Dynamic data representation
Type: byte, short, int, dict, list,…….
Attr: P/C,…….


8 bit
16 bit
Type
Length
Value
Dynamic Event
Type: 1 ----- variable name
2 ----- variable addr
3 ----- variable value
Length: the length of current type