Algorithm 1: Retrieve all memory reads and writes for a VEX IRSB I, using successor state S, producing A.

```
A, T ← ∅
foreach i ∈ I do
    if Type(i) = Store then
        if Type(i.addr) = Const then
            // Write to constant address
            A ← A ∪ i.addr
        end
        else
            // Write to variable address
            T ← T ∪ i.addr
        end
    end
    if Type(i) = WrTmp ∧ Type(i.data) = Load then
        if Type(i.data.addr) = Const then
            // Read from constant address
            A ← A ∪ i.data.addr
        end
        else
            // Read from variable address
            T ← T ∪ i.data.addr
        end
    end
end
// Use S to avoid recomputing ASTs
foreach t ∈ T do
    A ← A∪ EvalTmp(S,t)
end
```

Algorithm 2: Detect stepping behavior in a sequence of states S, iterating a loop. `IsTmpStore` is true when the VEX IRSB instruction is a `WrTmp` and its expression is `Store`.

```
R ← False
I ← ∅
foreach s ∈ S do
    foreach i ∈ s.irsb.statements do
        if IsTmpStore(i) then
            I[i.addr] ← I[i.addr] ∪ i.addr.tmp
        end
    end
end
foreach a ∈ I do
    l ← I[a].size
    if l > 1 then
        if I[a][0] ≤ I[a][1] ≤ ... ≤ I[a][l] then
            R ← True
        end
        if I[a][0] ≥ I[a][1] ≥ ... ≥ I[a][l] then
            R ← True
        end
    end
end
```

Algorithm 3: Tainting algorithm to obtain the registers and addresses used to calculate a VEX IR temporary variable.

```
Input: VEX IR statements S starting from last executed.
Tmp n to taint initially.
Result: Addresses A and registers R used to calculate n.
A, R ← ∅
T ← {n}
foreach s in S do
    if Type(s) = Put and Type(s.data) = RdTmp then
        if s.data.tmp ∈ T then
            R ← R ∪ {s.register}
        end
    end
    if Type(s) = WrTmp and s.tmp ∈ T then
        foreach a in s.data.args do
            if Type(a) = Get then
                R ← R ∪ {a.register}
            end
            if Type(a) = RdTmp then
                T ← T ∪ {a.tmp}
            end
            if Type(a) = Load then
                A ← A ∪ EvalTmp(a.address)
            end
        end
    end
end
```