A Original IDE Algorithm

Algorithm 4 Original IDE algorithm from [21]. Shortened version of the ForwardComputeJumpFunctionsSLRPs procedure.

```
while $PathWorkList \neq \emptyset$ do
    Select and remove an item $\langle s_p, d_1 \rangle \rightarrow \langle n, d_2 \rangle$ from $PathWorkList$;
    $f \longleftarrow JumpFn(\langle s_p, d_1 \rangle \rightarrow \langle n, d_2 \rangle)$;
    switch $n$ do
        case $n$ is call node in $p$ calling a procedure $q$ do
            foreach $d_3$ s.t. $\langle n, d_2 \rangle \rightarrow \langle s_q, d_3 \rangle \in E^{\#}$ do
                Propagate($\langle s_q, d_3 \rangle \rightarrow \langle s_q, d_3 \rangle, \lambda x.x$);
            $r \longleftarrow$ return-site node for $n$;
            foreach $d_3$ s.t. $e = \langle n, d_2 \rangle \rightarrow \langle r, d_3 \rangle \in E^{\#}$ do
                Propagate($\langle s_p, d_1 \rangle \rightarrow \langle r, d_3 \rangle, EdgeFn(e) \circ f$);
            foreach $d_3$ s.t. $f_3 = SummaryFn(\langle n, d_2 \rangle \rightarrow \langle r, d_3 \rangle) \neq \lambda x.\top$ do
                Propagate($\langle s_p, d_1 \rangle \rightarrow \langle r, d_3 \rangle, f_3 \circ f$);
        case $n$ is exit node of $p$ do
            foreach call node $c$ calling $p$ do
                $r \longleftarrow$ return-site for $c$;
                foreach $d_4, d_5$ s.t. $\langle c, d_4 \rangle \rightarrow \langle s_p, d_1 \rangle \in E^{\#} \wedge \langle e_p, d_2 \rangle \rightarrow \langle r, d_5 \rangle \in E^{\#}$ do
                    ...
                    if $f' \neq SummaryFn(\langle c, d_4 \rangle \rightarrow \langle r, d_5 \rangle)$ then
                        $SummaryFn(\langle c, d_4 \rangle \rightarrow \langle r, d_5 \rangle) \longleftarrow f'$;
                        $s_q \longleftarrow$ start node of $c$'s procedure;
                        foreach $d_3$ s.t. $f_3 = JumpFn(\langle s_q, d_3 \rangle \rightarrow \langle c, d_4 \rangle) \neq \lambda x.\top$ do
                            Propagate($\langle s_q, d_3 \rangle \rightarrow \langle r, d_5 \rangle, f' \circ f_3$);
        otherwise do
            foreach $m, d_3$ s.t. $\langle n, d_2 \rangle \rightarrow \langle m, d_3 \rangle \in E^{\#}$ do
                Propagate($\langle s_p, d_1 \rangle \rightarrow \langle m, d_3 \rangle, EdgeFn(\langle n, d_2 \rangle \rightarrow \langle m, d_3 \rangle) \circ f$);
end
```

Algorithm 5 The Propagate procedure from the original IDE algorithm [21].

```
Procedure Propagate($e, f$)
    let $f' = f \sqcap JumpFn(e)$;
    if $f' \neq JumpFn(e)$ then
        $JumpFn(e) := f'$;
        Insert $e$ into $PathWorkList$;
```

Algorithm 6 Phase II of the original IDE algorithm [21]

```
Procedure ComputeValues()
    // Phase II.i (value propagation)
    foreach $(n, d) \in N \times D$ do $val(n, d) \longleftarrow \top$;
    $val(s_{main}, \Lambda) \longleftarrow \bot$;
    $NodeWorkList \longleftarrow \{(s_{main}, \Lambda)\}$;
    while $NodeWorkList \neq \emptyset$ do
        Select and remove an ESG node $(n, d)$ from $NodeWorkList$;
        switch $n$ do
            case $n$ is start node of $p$ do
                foreach call node $c$ inside $p$ do
                    foreach $d'$ s.t. $f' = JumpFn(\langle n, d\rangle \rightarrow \langle c, d'\rangle) \neq \lambda\ell.\top$ do
                        PropagateValue($c, d', f'(val(s_p, d))$);
            case $n$ is call node in $p$, calling $q$ do
                foreach $d'$ s.t. $\langle n, d\rangle \rightarrow \langle s_q, d'\rangle \in E^{\#}$ do
                    PropagateValue($s_q, d', EdgeFn(\langle n, d\rangle \rightarrow \langle s_q, d'\rangle)(val(n, d))$);
    // Phase II.ii (value computation)
    foreach node $n$ in procedure $p$, that is not call or start node do
        foreach $d, d'$ s.t. $f' = JumpFn(\langle s_p, d'\rangle \rightarrow \langle n, d\rangle) \neq \lambda\ell.\top$ do
            $val(n, d) \longleftarrow val(n, d) \sqcap f'(val(s_p, d'))$;
```

Algorithm 7 The PropagateValue procedure from the original IDE algorithm [21]

```
Procedure PropagateValue($n, d, v$)
    $v' \longleftarrow v \sqcap val(n, d)$;
    if $v' \neq val(n, d)$ then
        $val(n, d) \longleftarrow v'$;
        Insert $(n, d)$ into $NodeWorkList$;
```