1 Appendix

Below is the proof of Theorem 1. In the proof, we refer to a sequence of transitions in the global transition graph as a *path*; an execution is a path starting with the initial global state. We write a path as $x_0; a_0; x_1; a_1; \ldots; x_n$ where for each i, the x_i are global states, the a_i are elements of $\Sigma \cup \{\epsilon\}$, and each triple (x_i, a_i, x_{i+1}) on a path is a valid transition. The length of a path is the number of transitions on it (n for the path above). For a path as above, we let $a_0; a_1; \ldots; a_{n-1}$ be the trace of actions along the path. Since an automaton runs on such traces, we can merge a run $q_0; a_0; q_1; a_1; \ldots; q_n$ of the automaton along this trace into the path itself; thinking of it as a run of the automaton on the path. This is written as $(x_0, q_0); a_0; (x_1, q_1); a_1; \ldots; (x_n, q_n)$.

Lemma 1 *For each configuration (f, n, p, q) that is generated by the algorithm, there is a program execution to a global state of the form $x(f, n)$, for some x, and a run of the automaton on this execution that is in state q at the end, and in state p at the last call to f.*

Proof. The proof is by induction on the number of algorithm steps. We strengthen this claim by adding the following: for any function f, and every pair $\langle q, q' \rangle$ in the summary for f, there is a path π starting with a global state $x(f, \downarrow)$ and ending at state $x(f, \uparrow)$ (for some x) and a run of the automaton on this path starting at state q and ending at state q'.

Basis (0 steps): the claim is true by the initialization of the workset, and as all summaries are initially empty.

Induction (k+1 steps): assume that the claim holds after k steps and consider the configurations and summaries generated in the $k+1$'st call to the STEP function. Let $c = (f, n, p, q)$ be the configuration examined at this stage. By the IH, there is an execution π whose final state has the form $x(f, n)$, and a run ρ on π where q is the end state and p is the state at the last call to f on π.

(i) if $n = \uparrow$, the pair $\langle p, q \rangle$ is added to the summary for f. The suffix of ρ beginning at the last call to f defines a path and a run on it from p to q, which validates the IH for the new summary pair. The call sites added to the workset are not newly generated configurations, so the IH holds for those.

(ii) now suppose that $n \neq \uparrow$. For a configuration generated in following an edge $e = (n, a, n')$ from n, there are two possibilities. Write ρ as $\rho'; a; (x(f, n), q)$.

(a) $a = \mathit{call}(f')$. Two types of configurations are generated. Firstly, the configuration $(f', \downarrow, q, q)$. Consider the path $\pi; \epsilon; (x(f, n')(f', \downarrow))$ defined by rule 2. The run $\rho'; a; (x(f, n), q); \epsilon; (x(f, n')(f', \downarrow), q)$ over this path justifies the new configuration.

In addition, a configuration (f, n', p, q') might be added, where $\langle q, q' \rangle$ is a summary pair for f'. By IH, the summary pair is associated with a run of the form $(y(f', \downarrow), q); \sigma[y]; (y(f', \uparrow), q')$, for some y, where each global state in $\sigma[y]$ has prefix y. The stack nature of the global transition graph implies that this input-output behavior is actually valid for any y. Consider the run obtained from ρ

where f' is called, does the sequence of actions specified in the trace for σ, and returns to f. $\rho; \epsilon; (x(f,n')(f',\downarrow), q); \sigma[x(f,n')]; (x(f,n')(f',\uparrow), q'); \epsilon; (x(f,n'), q')$. This run justifies the configuration (f, n', p, q').

(b) a is a non-call edge. Then a configuration (f, n', p, q') can be added, where $q' \in \Delta(q, a)$. Extend the run ρ to $\rho; a; (x(f,n'), q')$, which justifies the new configuration.

Lemma 2 *If there is an execution with final state $x(f,n)$, for some x, and a run of the automaton on this path that is in state q at the end, and in state p at the last call to f, then the configuration (f, n, p, q) is generated by the algorithm.*

Proof. The proof is by induction on the number of transitions along the execution path.

Basis (0): The final state of the execution is an initial global state, so the claim holds by the initialization of the worklist.

Induction $(k+1)$: Suppose that the claim holds for executions of length k. Consider a run on an execution of length $k+1$. This has the form $((main, \downarrow), q_0); \ldots; (x(f,n), q); a; (y, q')$, where q_0 is an initial automaton state. By the IH, the configuration (f, n, p, q) is generated at some point by the algorithm, where p is the automaton state at the last call to f. We have to show that a configuration appropriate to (y, q') is generated as well. There are three possibilities.

(i) a is not a call or return transition, so that $y = x(f, n')$, and $q' \in \Delta(q, a)$. When (f, n, p, q) is processed, this case leads to the creation of (f, n', p, q').

(ii) a is a call transition, so that $y = x(f,n)(f', \downarrow)$ and $q' = q$. The processing of the transition creates the configuration $(f', \downarrow, q, q)$, as appropriate.

(iii) a is a return step, so that $n = \uparrow$, $q' = q$ and $y = x$. I.e., the final state of the run is (x, q). By the IH, the configuration $(f, \uparrow, p, q)$ must be generated. Hence, a summary pair $\langle p, q \rangle$ is added to the summary for f when this configuration is processed. Suppose that x has the form $x'(f', n')$. We have to show that (f', n', p', q), for p' being the automaton state at the last call to f', is generated. Suppose that $x'(f', m)$ is the global state at the start of the last call to f. By the IH, the configuration (f', m, p', p) must be generated at some stage. Now there are two possibilities: if the summary pair $\langle p, q \rangle$ exists at this stage, the FOLLOW-EDGE code ensures that (f', n', p', q) is generated. If not, then (f', m, p', p) is added to the call sites for f. When the summary pair is generated later for f, this configuration is put back on the worklist and the summary pair is available when it is re-processed.

Theorem 1 *The algorithm reports an error at a configuration (f, n, p, q), for some p, q, if and only if there is a program execution ending at a global state with form $x(f,n)$, (for some x) labeled with trace t, such that the automaton has a rejecting run on t.*

Proof. (left-to-right) Suppose that the algorithm reports an error at (f, n, p, q). By Lemma 1, there is a program execution to a global state of the form $x(f,n)$ (for some x), and a run of the automaton on the associated trace that ends in

automaton state q. Since the procedure reports an error, q must be a rejecting automaton state, hence this is a rejecting run.

(right-to-left) Suppose that there is a program execution γ ending at a global state of the form $x(f, n)$ (for some x), labeled with trace t such that the automaton has a rejecting run on t. Let q be the final state of the automaton run, and let p be the state of the automaton corresponding to the last call to f on γ. Then, by Lemma 2, the configuration (f, n, p, q) must be generated by the procedure, and as q is a reject state, the procedure will report an error at this configuration.

Theorem 2 *The derivation graph resulting from the IncrFwd algorithm is exactly the graph generated by the Full analysis algorithm on the modified CFGs.*

Proof. (Sketch) Given the similarity between the algorithms, it is easy to show that any schedule of choices (and markings) from the workset in the checking algorithm corresponds with a schedule of choices (and generation) from the workset in the full analysis algorithm, and vice versa.

Precisely, one can show a bisimulation between the states of the Full and IncrFwd algorithm. In the bisimulation, algorithm states are related if (i) worksets are identical, (ii) for each function f, the summaries generated for f in Full are identical to the marked summaries for IncrFwd, and (iii) states visited in Full are exactly those marked in IncrFwd.

Theorem 3 *The derivation graph after the IncrBack algorithm is an over-approximation of the graph generated by the full analysis algorithm on the modified CFGs, but has precisely the same set of error configurations.*

Proof. The proof is based on reverse induction on the depth of SCC's in the (acyclic) SCC decomposition graph. It uses the following stronger induction hypothesis: at each stage, the repaired derivation graph and summary pairs for functions in SCCs below the current SCC are identical to those generated by applying the full analysis algorithm with the modified CFGs and the old entry configurations.

Consider an SCC with N levels. The IH is true initially as there are no SCCs at level $N + 1$. Consider the processing of an SCC C at level k. Functions in C can only access other functions in SCC's at levels k or higher. From the proof of Theorem 2, we claim that the checking process is equivalent to the generation process *from the given entry configurations* for functions in C. The generation and checking processes both make use of summaries for functions in SCC's below C, but these are accurate by the induction hypothesis. Hence, the summaries and derivation graphs produced for functions in C are also accurate.

The *main* function is at level 0. Applying the induction hypothesis to this SCC shows the first part of the claim. Thus, there can only be more error configurations than those generated by the full algorithm, and the excess configurations can be traced back to old entry configurations that are now unreachable. This situation is detected by RETRACE. After this final step, the error configurations are exactly those determined by the full analysis.