Using postdomination to reduce space requirements of data flow analysis

C. Bernardeschi *, G. Lettieri, L. Martini, P. Masci

Dip. di Ingegneria dell’Informazione, Università di Pisa, Via Diotisalvi 2, 56126 Pisa, Italy

Received 3 May 2005; received in revised form 16 November 2005
Available online 27 December 2005
Communicated by A.A. Bertossi

Keywords: Algorithms; Data flow analysis; Proof system

1. Introduction

Data Flow Analysis (DFA for short) is a basic technique to collect statical information on run-time behaviors of programs: it is essential in optimizing compilers and is also used in type inference problems [1]. When performing a DFA, the program to be analyzed is modeled by its Control Flow Graph (CFG) and by a set of transformation functions, one for each node in the graph. The set of program execution states is generally abstracted in a lattice. Each function models the effect of execution of the corresponding program instruction on the (abstracted) program execution state. Then, starting from the initial node in the CFG, and from an abstraction of the initial execution state, every possible execution path in the CFG should be tried, and the execution states encountered at each node collected. Standard DFA uses a fix point iteration to find an approximation of this information. DFA is expensive in time (because of the fixed point iteration), but also in space: a representation of the program execution state must be stored for each CFG node for the entire duration of the analysis. In this paper we propose a method to reduce space usage during some kind of DFA analysis. The method proceeds as follows: first, we define a formal proof system for sentences like “all execution paths that start at node \( n \), with initial execution state \( f \), and reach node \( m \), have been tried”; the rules in the proof deduce sentences about sets of paths, based on the validity of sentences on sub-paths. We show that every proof in our system contains an approximation of the information collected at nodes when all paths are executed. Then, we give an algorithm to find proofs in our system. The algorithm constructs the proof in a goal directed (or backward) fashion. The algorithm has the potential to save space since it can release some program state information each time it has constructed a proof for a sub-goal. This space saving is possible, for example, when we are interested in the (non)existence of a given execution state or in the collection of information only at some program points. In particular, Java bytecode verification fits these requirements, and the proposed algorithm can be used to develop a Verifier on memory limited embedded systems, e.g., Java cards [5].

2. Basic concepts

Definition 1 (Directed graph). A directed graph \( G = (V, E) \) consists of a set of nodes \( V \) together with a set of directed edges \( E \subseteq V \times V \).
The function \( \text{succ}: V \rightarrow 2^V \) returns the successors of a node: \( \text{succ}(n) = \{ m \mid (n, m) \in E \} \).

**Definition 2 (Path).** A path from \( n \) to \( m \) is a sequence of nodes \( n = n_1, \ldots, n_j = m \) such that \( (n_i, n_{i+1}) \in E \) for \( 1 \leq i < j \).

Note that the null path \( n \) is always a valid path from \( n \) to \( n \). In the following, to denote a path \( n_1, \ldots, n_j \) we will use the string \( n_1 \cdots n_j \). We write \( n \in p \) if \( p = p_1, \ldots, p_k \) for some possibly empty strings \( p_1, \ldots, p_k \).

Given a graph \( G = (V, E) \) and a set of nodes \( K \subseteq V \), \( G \ominus K \) is the subgraph of \( G \) obtained by pruning the edges from each \( n \in K \) to its successors:

\[
G \ominus K = (V, E / \{ (n, m) \mid n \in K \}).
\]

The set of all paths from node \( n \) to node \( m \) in \( G \) is denoted as \( n \rightarrow^* G m \). We use the notation \( n \xrightarrow{K} m \) instead of \( n \rightarrow^* G \ominus K m \), when \( G \) is clear from the context. Moreover, when \( K = \emptyset \) we write \( n \Rightarrow m \) and when \( K = \{ v \} \) we write \( n \Rightarrow v \).

The CFG of a program is a directed graph, containing the control dependences among the instructions of the program [1]. Each node in the CFG represents an instruction or a basic block.

**Definition 3 (CFG).** A CFG is a 4-tuple \( (V, E, \text{start}, \text{end}) \) where \( (V, E) \) is a directed graph, \( \text{start}, \text{end} \in V \), \( \text{succ} (\text{end}) = \emptyset \) and \( \forall v \in V \), there is a path from start to \( v \) and from \( v \) to end.

An example of control flow graph is shown in Fig. 1(a), where start is node 1 and end is node 5.

The immediate postdominator of a node \( n \) in a directed graph represents the first common node in all the paths that start at node \( n \).

**Definition 4 (Postdomination).** Let \( n,m \in V \): \( m \) postdominates \( n \), denoted by \( m \xrightarrow{\text{pd}} n \), if \( m \) is different from \( n \) and \( m \) is on every path from \( n \) to \( m \). Moreover \( m \) immediately postdominates \( n \), denoted by \( m = \text{ipd}(n) \), if \( m \xrightarrow{\text{pd}} n \) and there is no node \( r \) such that \( m \xrightarrow{\text{pd}} r \xrightarrow{\text{pd}} n \). We use \( \text{ipd} \) to indicate the reflexive closure of \( \text{pd} \), that is \( m \xrightarrow{\text{ipd}} n \iff m \xrightarrow{\text{pd}} n \) or \( m = n \).

For example, in Fig. 1(a), \( \text{ipd}(1) = \text{ipd}(2) = \text{ipd}(4) = 5 \) and \( \text{ipd}(3) = 2 \). Since end is reachable from any other node of the CFG, the immediate postdominator always exists for each node different from end. Both the \( \text{pd} \) and the \( \text{pd} \) relations are transitive. Moreover, \( \text{pd} \) is asymmetric. The following theorem\(^1\) shows that \( \text{ipd}(n) \) is on every path from \( n \) to any node \( m \) such that \( m \xrightarrow{\text{pd}} n \).

**Theorem 1.** Given \( n,m \in V \), if \( m \xrightarrow{\text{pd}} n \), then \( m = \text{ipd}(n) \lor (n \xrightarrow{\text{ipd}(n)} m = \emptyset) \).

### 2.1. Collecting abstract execution states

DFA calculates a representation of the collecting semantics of an abstract interpretation of a program [3]. For our purposes, we can define abstract interpretation and collecting semantics in the following way. We assume a finite complete lattice \( (F, \sqsubseteq) \) ranged over by \( f, f', g, \ldots \), partially ordered by \( \sqsubseteq \). Given \( f, g \in F \), \( f \sqsubseteq g \) means \( f \sqsubseteq g \) and \( f \neq g \). Moreover, \( f \sqcup g \) denotes the least upper bound of \( f \) and \( g \). We use \( \bot \) to denote the least element in \( F \). Given a CFG \( (V, E, \text{start}, \text{end}) \), an abstract execution state is denoted as \( n : f \), with \( n \in V \) and \( f \in F \), where \( f \) represents the per-node information needed by the analysis (frame, from now on). For each node \( n \) in the CFG, we introduce a transformation function \( I_n : F \rightarrow F \). The transformation function models the effect of the execution of the instruction at node \( n \) on the frame part of the abstract execution state. Given an abstract execution state \( n : f \), the abstract interpretation of the CFG produces a set of “next” abstract execution states, given by \( m : I_n(f) \), for each \( m \in \text{succ}(n) \). This gives an abstract execution tree, rooted at \( \text{start} : f_0 \), where \( f_0 \) models the initial program.

---

1 All proofs omitted in this paper can be found in [2].
state. We assume that \( \forall n, I_n \) is a monotonic function. Given a path \( p = n_1 \ldots n_k \), we denote by \( I_p \) the composition of the transformation functions of the path: \( \forall f, I_p(f) = I_n(\ldots(I_{n_1}(f)\ldots)) \). If \( p \) is empty, \( I_p \) is the identity function.

The collecting semantics associates to each node of the CFG the set of frames that appeared at the node in the abstract execution tree. We are interested in all frames that appear in paths that start at a given node, and end in any node belonging to a given set \( K \). To this end, we introduce the following definition:

\[ \text{Definition 5} \ (N^n_K^{f}). \text{ Let } G = (V, E) \text{ and } K \subseteq V. \]
\[ N^n_K^{f} \subseteq V \times F \text{ is defined as the set of pairs such that: } m : g \in N^n_K^{f} \iff \exists pm \in n \quad m \text{ and } g = I_p(f). \]

Note that \( N^n_0^{\text{start}} : f_0 \) is the collecting semantics of the program. The following lemma can be easily proved:

\[ \text{Lemma 1.} \text{ Let } K_1, K_2 \subseteq V, n \in V, f \in F. \text{ If } K_1 \subseteq K_2 \text{ then } N^n_{K_2}^{f} \subseteq N^n_{K_1}^{f}. \]

The following relation is defined on sets of abstract states:

\[ \text{Definition 6} \ (\subseteq). \text{ Given } A, B \subseteq V \times F, A \subseteq B \iff \forall n : g \in A, \exists n : g' \in B \text{ such that } g \subseteq g'. \]

Note that \( A \subseteq B \) implies \( A \preceq B \). The following lemma can be proved by monotonicity of \( I_n \).

\[ \text{Lemma 2.} \text{ Let } K \subseteq V, n \in V \text{ and } f, f' \in F \text{ with } f \subseteq f', \text{ it is } N^n_K^{f'} \subseteq N^n_K^{f}. \]

We use \( \downarrow_V \) to denote the projection of a set of node-frame pairs onto the set of nodes, i.e., given \( N \subseteq V \times F \), \( N \downarrow_V = \{ n \in V \mid \exists f \in F, n : f \in N \} \).

### 3. A formal proof system

In this section we give a formal proof system for sentences of the form \( \Gamma \mid n : f \rightarrow m : g \), where \( \Gamma : V \rightarrow F \) represents the constraints, \( n \in V \) is the source node, \( f \in F \) is the source frame, \( m \in V \) is the target node, and \( g \in F \) is the target frame. Assume first that the constraints are empty (\( \text{dom}(\Gamma) = \emptyset \)). In this scenario, this sentence means that, in all execution paths that start with execution state \( n : f \), and end in node \( m \), the execution arrives to node \( m \) with a frame \( g' \subseteq g \). If the constraints are not empty, the sentence has the same meaning, as before, for the paths that never reach a node in \( \text{dom}(\Gamma) \). For the paths that reach a node \( n' \in \text{dom}(\Gamma) \), instead, the sentence says that, the first time node \( n' \) is reached, the execution state is \( n' : f' \), with \( f' \subseteq \Gamma(n') \). In this case, nothing is said on the rest of the path (including the target frame).

Fig. 2 reports a formal proof system for sentences. In the rules, the notation \( \Gamma, n : f (n \notin \text{dom}(\Gamma)) \) represents the function \( \Gamma' \) such that \( \Gamma'(n) = f \), and \( \Gamma'(n') = \Gamma(n'), \forall n' \in \text{dom}(\Gamma) \). Rule \text{NULL} considers the case when the target node is the same as the source node. In this case, there is only one path to consider: the null path that starts and ends in the source node. Here the sentence is clearly true if the target frame is equal to the source frame. Rule \text{CYCLE} says that, if the source node is in the constraints, the sentence is true, provided that the source frame verifies the constraint. Note that the soundness of the formal proof system could be preserved by choosing, in each rule, target frames greater than, or equal to, the ones given in Fig. 2. The rules are given in the present form to increase the accuracy (i.e., to prove Theorem 6). Rule \text{IPD} considers the case when the target node is the immediate postdominator of the source node. The rule says that the sentence \( s \) is true if there exists a frame \( f'' \) such that all the premises are true. Each premise is a new sentence, where the source node is a successor of the source node of \( s \), and the target node is the same as in \( s \). In each premise,
a constraint of $f''$ on the source node of $s$ is added. Rule PD considers the case when the target node post-
odates the source node, but it is not the ipd of the source node. The rule says that the sentence is true if we can prove a similar sentence on all the paths from the source node to its ipd, and then from the ipd to the target node.

A derivation tree is a tree of sentences with leaves at the top and a root at the bottom, where each sentence is obtained from the ones immediately above it by some rules of the proof system. A sentence is derivable if it is the root of a derivation. Note that a given sentence may be the root of more than one derivation. In particular, there may be different premises that meet the requirements of rule IPD, each with a different value for $f''$. Each derivation tree $T$ has the form:

$$T = \frac{T_1 \cdots T_s}{\Gamma \mid n: f \to m: f''},$$

where $T_1, \ldots, T_s$ are (possibly empty) derivation trees. We use $\bar{T}$ to indicate the root of the tree (the conclusion sentence). We use $[T]$ to indicates the set of derivations that are a proof for the premise sentences of $\bar{T}$. It is:

$$\bar{T} = \Gamma \mid n: f \to m: f'' \text{ and } [T] = \{T_1, \ldots, T_s\}.$$  

We are interested in properties of entire derivations, rather than in properties of derivable sentences. We write $\|T\|$ to indicate the set of pairs $n: f$, such that $n: f$ appears as the source of at least one sentence of $T$. The set $\|T\|$ can be recursively defined as follows:

(i) $\| \| = \emptyset,$

(ii) given $T$ with $\bar{T} = \Gamma \mid n: f \to m: f'$, $\|T\| = \bigcup_{T' \in [T]} \|T'\| \cup \{n: f\}.$

The following lemma states that the frame $\Gamma(n)$ in a derivable sentence is an upper bound for all the frames paired with the target node in all the proofs of the sentence.

**Lemma 3.** Given a derivation tree $T$, with $\bar{T} = \Gamma \mid n: f \to m: f'$, $\forall \xi \in \text{dom}(\Gamma), \text{ipd}(\xi) \not\subseteq m$, then $g \subseteq \Gamma(\xi).$

Some conditions must be satisfied by $\Gamma$ to ensure that a derivation tree does not contain more nodes than the corresponding collecting semantics.

**Lemma 4** (Confinement). Given a derivation tree $T$ with $\bar{T} = \Gamma \mid n: f \to m: f'$, if $\forall \xi \in \text{dom}(\Gamma), \text{ipd}(\xi) \not\subseteq m$, then $\|T\| \leq \|T\| \cup \{m: f\},$.

**Proof (Sketch).** By structural induction on the shape of the derivation. The hypothesis on $\Gamma$ is needed in the PD case, to prove that a node visited in the second sub-tree (whose source node is ipd($n$)) is indeed reachable from $n$ without visiting nodes in $\Gamma$. □

The following lemma states that the target frame of a derivable sentence is an upper bound of all the frames paired with the target node in all the proofs of the sentence.

**Lemma 5.** Given a derivation tree $T$ with $\bar{T} = \Gamma \mid n: f \to m: f''$ and $\forall \xi \in \text{dom}(\Gamma), \text{ipd}(\xi) \not\subseteq m$, then $g \subseteq \Gamma(f'').$

**Proof.** By structural induction on the shape of the derivation.

**NULL, CYCLE, IPD.** Trivial.

**PD.** It is $\bar{T} = \Gamma \mid n: f \to m: f''$ and $\{T\} = \{T_1, T_2\}$, with $T_1 = \Gamma \mid n: f \to \text{ipd}(n): f' \text{ and } T_2 = \Gamma \mid \text{ipd}(n): f' \to m: f''$. The induction hypotheses are: (1) if $\forall \xi \in \text{dom}(\Gamma), \text{ipd}(\xi) \not\subseteq \text{ipd}(n)$ and $\exists g \subseteq \|T_1\|$, then $g \subseteq f''$; (2) if $\forall \xi \in \text{dom}(\Gamma), \text{ipd}(\xi) \not\subseteq m$ and $\exists g \subseteq \|T_2\|$, then $g \subseteq f''$. The induction thesis is: if $\forall \xi \in \text{dom}(\Gamma), \text{ipd}(\xi) \not\subseteq m$ and $\exists g \subseteq \|T\|$, then $g \subseteq f''$. If the condition on $\text{dom}(\Gamma)$ in the induction thesis holds, then both conditions on $\text{dom}(\Gamma)$ in (1) and (2) hold as well. Consider now $m: g \subseteq \|T\|$. We can apply Lemma 4 to premise $T_1$, resulting in $\|T_1\| \cup V \subseteq N_{\text{dom}(\Gamma)}^{\text{ipd}(n)} \cup V$ and, since $m \not\subseteq \text{ipd}(n)$, it cannot be that $m: g \subseteq \|T_1\|$ (by Theorem 1), so we must have $m: g \subseteq \|T_2\|$, implying $g \subseteq f''$ by (2). □

The following theorem shows that every proof for a given sentence $s$ contains a safe approximation of the collection of abstract execution states obtained starting from the source of $s$, and ending in the nodes that are either a constraint, or the target of $s$.

**Theorem 2** (Safety). Given a derivation tree $T$ with $\bar{T} = \Gamma \mid n: f \to m: f'$, such that $\forall \xi \in \text{dom}(\Gamma), \text{ipd}(\xi) \not\subseteq m$, then $N_{\text{dom}(\Gamma)}^{\text{ipd}(n)} \cup \{m: f\} \subseteq \|T\|.$

**Proof.** By structural induction on the shape of the derivation.

**NULL, CYCLE.** Trivial.

**IPD.** Let $K' = \text{dom}(\Gamma) \cup \{\text{ipd}(n)\}$ and $K'' = K' \cup \{n\}$. Assume $\text{succ}(n) = \{m_1, \ldots, m_s\}$. It is $\forall T_i \in [T], \bar{T_i} = \Gamma \mid n: f'' \mid m_i: f_i \to \text{ipd}(n): f_i \text{ and } \bar{T} = \Gamma \mid n: f \to \text{ipd}(n): \bigcup_{1 \leq i \leq s} f_i$. The induction hypothesis is: $\forall i \in \{1, \ldots, s\}, N_{K''}^{m_i: f_i} \subseteq \|T_i\|$. The induction thesis
is: \( N^{n,f}_{K'} \leq ||T|| \). Consider \( \xi : g \in N^{n,f}_{K'} \). By Definition 5, \( \exists p \in n \xrightarrow{K'} \xi \), with \( p = q \xi \) and \( I_q(f) = g \). If \( q \) is empty, then it must be \( \xi = n \) and \( g = f \). Since \( n : f \) is the source of \( T \), \( n : f \in ||T|| \). If \( q \) is not empty, let \( p = np' \xi \) with \( p' \xi \in m_j \xrightarrow{K'} \xi \) for some \( m_j \in {\text{succ}}(n) \). Let \( p' = p'_1 p'_2 \cdots p'_{h-1} p'_h \) with \( p'_i \in m_{j_i} \xrightarrow{K'} n \), for some \( m_{j_i} \in {\text{succ}}(n) \); \( p' \in m_j \xrightarrow{K'} n \). Consider the first sub-path. By Definition 5 we can write: \( n : f' \in \xi \) (by Lemma 1). Consider now the second sub-path. Now \( n : I_{p'_1}(I_n(f')) \in N^{m_{j_1}:I_n(f')}_{K''} \). Since \( f'_1 \in f'' \), by the result of the first sub-path, \( N^{m_{j_1}:I_n(f')}_{K''} \leq N^{m_{j_1}:I_n(f')}_{K''} \leq ||T_j|| \) by induction hypothesis. Let \( f'_2 = I_{p'_2}(I_n(f'_1)) \). By Lemma 3, \( f'_2 \subseteq f'' \). The same reasoning applies to all sub-paths until \( p'_h \), and we obtain \( f'_h \subseteq f'' \). In the last path we have \( g = I_{p'_h}(I_n(f'_h)) \). It is \( \xi : g \in N^{m_{j_h}:I_n(f'_h)}_{K''} \). By induction hypothesis, \( N^{m_{j_h}:I_n(f'_h)}_{K''} \leq ||T_{j_{h-1}}|| \leq ||T|| \).

PD. Let \( K' = \text{dom}(\Gamma) \cup \{\text{ipd}(n)\} \) and \( K'' = \text{dom}(\Gamma) \cup \{m\} \). It is \( T = \Gamma | n : f \rightarrow m : f'' \) \( \uparrow \) and \( |T| = \{T_1, T_2\} \), with \( T_1 = \Gamma | n : f \rightarrow \text{ipd}(n) : f' \) and \( T_2 = \Gamma | \text{ipd}(n) : f' \rightarrow m : f'' \). The induction hypothesis is: \( N^{n,f}_{K'} \leq ||T_1|| \) and \( \text{ipd}(n) : f' \leq ||T_2|| \). The induction thesis is: \( N^{n,f}_{K''} \leq ||T|| \). Consider \( \xi : g \in N^{n,f}_{K''} \). By Definition 5, \( \exists p \in n \xrightarrow{K''} \xi \), with \( p = q \xi \) and \( I_q(f) = g \). We have two cases.

Case 1. ipd(\( n \) \( \notin \) \( p \); then, \( p \in n \xrightarrow{K''} \xi \). By Definition 5, it is \( \xi : I_q(f) \in N^{n,f}_{K''} \subseteq N^{n,f}_{K''} \) (by Lemma 1). By induction hypothesis, \( N^{n,f}_{K''} \leq ||T_1|| \leq ||T|| \).

Case 2. ipd(\( n \) \( \in \) \( p \); \( n \in K'' \). By Definition 5, it is \( \xi : I_q(f) \in N^{n,f}_{K''} \subseteq N^{n,f}_{K''} \) (by Lemma 1). By induction hypothesis, \( N^{n,f}_{K''} \leq ||T_1|| \leq ||T|| \).

4. The algorithm

In the following we give an algorithm that constructs a proof in the formal proof system in Fig. 2. The algorithm is given in two steps. Firstly, we give an algorithm to explore paths in the CFG, disregarding frames. Then, we describe a procedure that annotates nodes with frames. In the first step, paths are encoded in a tree structure, called the control tree, defined operatively as the result of the recursive procedure \( \text{ctree}(\gamma, n, m) \) shown in Fig. 3, where \( \gamma \subseteq V \) and \( n, m \in V \). The control tree for the CFG of Fig. 1(a) is shown in Fig. 1(b). The control tree relates sets of paths to other (smaller) sets of paths. The relation among sets of paths in the control tree is similar to the relation among sentences in a derivation in the proof system of Fig. 2, as far as the CFG nodes are concerned. The paths from a node \( n \) to a node \( m \) that postdominates \( n \) are split in the paths from node \( n \) to its ipd, joined to the paths that go from ipd(\( n \)) to \( m \). The paths that go from a node \( n \) to its ipd are the union of the paths that go from every successor of \( n \) to ipd(\( n \)). The set \( \gamma \) is used to break cycles, and plays a role similar to \( \text{dom}(\Gamma) \) in the formal proof system.

The following theorem states that the control tree for the paths that go from a node \( n \) to a node \( m \) always exists, provided that \( m \notin n \).

**Theorem 3** (Existence of control tree). Let \( n, m \in V \) such that \( m \notin n \) \( \gamma \subseteq V \):
(i) the execution of $ctree(γ, n, m)$ never reaches the statement fail;
(ii) $ctree(γ, n, m)$ terminates in a finite number of steps.

Proof. (i) By contradiction, assume a recursive call of $ctree$, with parameters $n'$, $m'$ and $γ'$ executes the fail instruction. This is only possible if $m' \not\in \overline{pd} n'$, but this cannot happen (by inspection of recursive calls in Fig. 3).

(ii) By contradiction. Let $s = s_1 s_2 \ldots$ be an infinite sequence of recursive invocations of $ctree(γ, n, m)$, such that $s_{i+1}$ is called by $s_i$. First we note $γ$ may never decrease in such a sequence. Therefore $s$ cannot contain an infinite number of invocations of type (*i) in $s$, since, at each invocation, $γ$ is augmented and, at most, it can contain all the nodes in $V$, which are finite. Let $s_k$ be the last invocation of type (*). Then $s' = s_{k+1}s_{k+2} \ldots$ only contains invocations of type (**i). Moreover, since the first of the two invocations in (**) will cause a subsequent invocation of type (*), then $s'$ only contains instances of the second invocation in (**i). Consider invocation $s_i$ in $s'$, with arguments $γ_i$, $n_i$ and $m_i$. Then, invocation $s_{i+1}$ will have arguments $γ_i$, ipd($n_i$) and $m_i$. Thus, the second argument of all recursive invocations in $s'$ form an infinite sequence $n_0, n_1, n_2, \ldots$ where $n_{i+1} = \text{ipd}(n_i)$. Since nodes are finite, at least a node $n$ must appear twice in this sequence. Suppose node $n$ appears at positions $j$ and $j + k$. We have $n = n_{j+k} \text{pd} n_{j+k-1} \text{pd} \ldots \text{pd} n_j = n$, implying $n \text{pd} n$ (by transitivity of pd) which is a contradiction. \hfill $\square$

Fig. 4 shows an algorithm to find a proof for the sentence $Γ \upharpoonright n : f \Rightarrow m : f''$. Given a proof $T$ we define tf($T$) as the target frame in $\overline{T}$. In Fig. 4, the statement “throw n : f” raises an exception of the form n : f.
An exception normally causes immediate termination of the current instance of $dfa$, and is rethrown in the enclosing instance, unless the exception is caught. The statement “do[c] while catch (n : f’’’’) catches all exceptions n : f’ thrown by command c. If an exception is caught, $f'$ is assigned to $f''$ and command c is reexecuted. Otherwise, the do statement terminates.

The algorithm constructs the proof in a goal oriented fashion, using a form of backtracking (embodied by the exception mechanism) to find a suitable frame $f''$ in the application of rule IPD. The following theorems show that the algorithm is correct.

Theorem 4 (Termination of $dfa$). Let $n, m \in V$, with $m \not\in \overline{pd} n$, $γ \subseteq V$, $t = ctree(γ, n, m)$, $Γ : γ \Rightarrow F$, $f \in F$, then: either $dfa(t, Γ, f)$ terminates correctly, or it throws $n : f$ such that $n \in γ$ and $Γ(n) \sqsubseteq f$.

Proof. The proof is by induction on the depth $d$ of $t$.

Base step. If $d = 1$, then it must be (by construction of $t$) that (i) $n = m$ and $n \notin γ$; (ii) $n \in γ$. In case (i), the algorithm will take branch (1) in Fig. 4, terminating correctly. In case (ii), it will take branch (2). There, it will either terminate correctly, or (if $f \not\subseteq Γ(n)$) it will throw $n : f$, with $n = n \in γ$ and $\overline{f} = f \cup Γ(n)$. Note that $Γ(n) \not\subseteq f$, and it cannot be $Γ(n) = f$, since this would mean that $f \subseteq Γ(n)$.

Induction step. Assume that the theorem holds for all control trees with depth $d'$ such that $1 \leq d' < d$. We want to show that the theorem also holds for all control trees with depth $d$. Since $d > 1$, we have two cases.

Case 1. $m = \text{ipd}(n)$: Branch (3) is taken. Each recursive invocation of $dfa$ will (by induction hypothesis) either terminate correctly or throw $n : f$, with $n \in γ \cup \{n\}$. If all recursive invocations terminate correctly, also $dfa

\[\text{dfa}\left(\frac{\text{do}\{\text{for } i \leq s \text{ do } T_i := \text{dfa}(t_i, (Γ, n : f''', I_n(f''')); \text{ while catch (n : f''');}}}{\text{return } Γ \upharpoonright n : f \Rightarrow m : tf(T_i)}\right)\]

Fig. 4. An algorithm for constructing a proof.
will terminate correctly. If a recursive invocation throws \( \tilde{n} : \tilde{f} \), we have two cases.

Case 1.1. \( \tilde{n} = n \): the exception is caught, and the do loop is repeated, with \( f'' = \tilde{f} \). Note that, by induction hypothesis, the new value of \( f'' \) is strictly greater than the previous one. Thus, this step cannot be repeated indefinitely, since lattice \( F \) is of finite height.

Case 1.2. \( \tilde{n} \neq n \): the exception is not caught. The current instance of \( dfa \) rethrows \( \tilde{n} : \tilde{f} \). The conditions on \( \tilde{n} \) and \( \tilde{f} \) are met by the induction hypothesis.

Case 2. \( m \neq \text{ipd}(n) \): Branch (4) is taken. By induction hypothesis, the two recursive invocations of \( dfa \) will either terminate correctly or throw \( \tilde{n} : \tilde{f} \). It is easy to see that, in both cases, the theorem holds.

Theorem 5. Let \( n, m \in V \), with \( m \neq \text{ipd}(n) \); \( \Gamma : \gamma \rightarrow F \), with \( \gamma \subseteq V \); \( f \in F \) such that \( f \subseteq \text{MOP}(n) \); \( I_n : F \rightarrow F \) be distributive over \( \sqcup \) for all \( n \in V \). Let \( t = \text{ctree}(\gamma, n, m) \):

(i) if \( dfa(t, \Gamma, f) \) terminates correctly, it returns \( T \) such that \( tf(T) \subseteq \text{MOP}(m) \) and \( \forall v : g \in ||T||, g \subseteq \text{MOP}(v) \);

(ii) if \( dfa(t, \Gamma, f) \) raises an exception \( v : g \), then \( g \subseteq \text{MOP}(v) \).

Proof. The proof is by induction on the depth \( d \) of \( t \).

Base step. If \( d = 1 \), then either branch (1) or (2) in Fig. 4 is taken. The step is trivially proved by inspection of Fig. 4.

Induction step. Assume that the theorem holds for all control trees with depth \( d' < d \), with \( d > 1 \). We prove that the theorem holds for all control trees with depth \( d \). Since \( d > 1 \), either branch (3) or (4) in Fig. 4 is taken. If branch (3) is taken, it is \( m = \text{ipd}(n) \), \( \tilde{T} = \Gamma \setminus n : f \mapsto \text{ipd}(n) : \bigcup_{1 \leq i \leq s} tf(T_i^1) \) and \( |T| = \{T_1', \ldots, T_s'\} \), where \( T_i' = dfa(t_i, (\Gamma, n : f), I_n(f'')) \), with depth of \( t_i < d \), and \( v_i \) is the source node of \( T_i' \). The first time the loop is executed, \( f'' = f \subseteq \text{MOP}(n) \) (by hypothesis). Moreover, \( I_n(f'') \subseteq \text{MOP}(v_i) \), by Lemma 6. Since the theorem holds on each recursive call (by induction hypothesis), and the hypotheses of the theorem are easily verified, we know that either point (i) or (ii) holds for each call. If a recursive call raises an exception \( v : g \) (point (ii)) we have two cases: if \( v \neq n \), the exception is rethrown, with \( g \subseteq \text{MOP}(v) \) and the induction step is proved; if \( v = n \), the exception is caught and the loop is repeated with \( f'' = g \subseteq \text{MOP}(n) \). The same reasoning can be repeated, until all recursive calls terminate correctly (we know by Theorem 4 that this will eventually occur). Then, we have \( tf(T_i^1) \subseteq \text{MOP}(\text{ipd}(n)) \) \((1 \leq i \leq s) \) and \( \bigcup_{1 \leq i \leq s} tf(T_i^1) \subseteq \text{MOP}(\text{ipd}(n)) \) (by definition of \( \sqcup \)). Finally, point (i) for the induction step is proved by observing that \( ||T|| = \bigcup_{1 \leq i \leq s} ||T_i^1|| \cup |n : f| \), \( f \subseteq \text{MOP}(n) \) by hypothesis, and point (i) already holds for each \( T_i' \). If branch (4) is taken, the induction step is easily proved by applying the theorem to the two recursive calls, in sequence.

The complete analysis is performed by \( dfa(\text{ctree}(\emptyset, \text{start}, \emptyset), \emptyset, f_0) \) that, by Theorem 4, always terminates correctly (no exception can be thrown, since \( \text{dom}(\Gamma) = \emptyset \)). Moreover by Theorem 2 every proof \( T \) is a safe approximation of the collection of abstract execution states and by Theorem 6 every node is visited with a frame \( \sqsubseteq \text{MOP}(n) \).

In embedded system, RAM is a severely limited resource. The \( dfa \) algorithm in Fig. 4 is able to reduce RAM usage in DFA analysis provided that: the CFG and the \text{ipd} function (which are read only information) are stored in PROM; the control tree is built...
on the fly; \( \Gamma \) is implemented with a global stack that grows/shrinks whenever branch (3) is taken/returns; \( dfa \) only returns the target frame of the conclusion of the sub-derivation. Note that branch (3) is non trivial (that is, it causes recursive calls to \( dfa \) that do not terminate immediately) when the source node represents a control instruction of the program (that is, it has more than one successor). In this case, branch (3) causes all paths that belongs to the “scope” of the control instruction to be tried, collecting information at the “control instruction exit” (that is, the point where all paths originating from the control instruction meet: the ipd). The maximum size of the \( \Gamma \) stack, then, is almost equal to the maximum “nesting depth” of control instructions of the program. The number of frames that must be stored in RAM can be estimated in twice the maximum depth of the global stack (for each element of the stack we need one frame for the element itself, and one frame for the accumulation at ipd targets). By contrast, we can think of standard DFA as requiring a number of frames roughly equal to the total number of control instructions. Since the maximum nesting depth is largely independent from the total number of control instructions, our approach has the potential to be feasible for a larger number of programs, for a given amount of RAM.

Acknowledgements

We wish to thank the anonymous referees for their constructive comments that have helped us to clarify and correct the presentation of this Letter.

References


