Beyond Labels: Permissiveness for Dynamic Information Flow Enforcement*

Elisavet Kozyri Fred B. Schneider
Department of Computer Science
Cornell University
{ekozyri,fbs}@cs.cornell.edu

Andrew Bedford Josée Desharnais Nadia Tawbi
Department of Computer Science
Laval University
{andrew.bedford.1@, josee.desharnais@ift., nadia.tawbi@ift.}ulaval.ca

February 28, 2019

Abstract

Flow-sensitive labels used by dynamic enforcement mechanisms might themselves encode sensitive information, which can leak. Meta-labels, employed to represent the sensitivity of labels, exhibit the same problem. This paper derives a new family of *enforcers*—*k-Enf*, for $2 \leq k \leq \infty$—that uses *label chains*, where each label defines the sensitivity of its predecessor. These enforcers satisfy *Block-safe Noninterference* (BNI), which proscribes leaks from observing variables, label chains, and blocked executions. Theorems in this paper characterize where longer label chains can improve the *permissiveness* of dynamic enforcement mechanisms that satisfy BNI. These theorems depend on semantic attributes—*k-precise*, *k-varying*, and *k-dependent*—of such mechanisms, as well as on initialization, threat model, and lattice size.

*Kozyri and Schneider are supported in part by AFOSR grant F9550-16-0250 and NSF grant 1642120. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of these organizations or the U.S. Government. Bedford, Desharnais, and Tawbi are supported by NSERC grants RGPIN-239294-2012 and RGPIN-04461-2015.

1 Introduction

Dynamic enforcement mechanisms for information flow control employ tags containing labels to represent the sensitivity[1] of what variables store. These labels can be *flow-sensitive*, meaning that they change when a value with different sensitivity is assigned to the tagged variable during program execution. Sensitive information might influence which assignments execute and, consequently, determine how and when the flow-sensitive label tagging a variable changes. So flow-sensitive labels can depend on sensitive information.

Inspecting or directly observing flow-sensitive labels might itself leak sensitive information [20]. Consider a program

$$\textbf{if } m > 0 \textbf{ then } w := h \textbf{ else } w := l \textbf{ end} \tag{1}$$

Suppose w is tagged with a flow-sensitive label, but the other variables are tagged with *fixed* labels, which do not change during execution: l is tagged with fixed label L (i.e., low), m with M (i.e., medium), and h with H (i.e., high), where $\mathsf{L} \sqsubset \mathsf{M} \sqsubset \mathsf{H}$ holds.

(i) If $m > 0$ holds, then information flows *explicitly* from h to w and *implicitly* from m (in $m > 0$) to w. When (1) terminates, w should be tagged with flow-sensitive label H, because H is at least as restrictive as the label H that tags h and the label M that tags m.

(ii) If $m \not> 0$ holds, then w should be tagged with flow-sensitive label M when (1) terminates, because M is at least as restrictive as the labels that tag l and m.

So, the flow-sensitive label tagging w depends on whether $m > 0$ holds. Information about m, which is sensitive, leaks to observers that can learn that label.

Blocking (e.g., [1, 15, 26, 33]) an execution based on flow-sensitive labels might leak sensitive information, too. Consider (1), extended with two assignments:

$$\begin{aligned}&\textbf{if } m > 0 \textbf{ then } w := h \textbf{ else } w := l \textbf{ end};\\ &m := w;\ l := 1\end{aligned} \tag{2}$$

[1] In this paper, *sensitivity* refers to *confidentiality level*.

(i) If $m > 0$ holds, then $m := w$ should be blocked to prevent information tagged H and stored in w from flowing to m; assignment $l := 1$ is not reached.

(ii) If $m \not> 0$ holds, then $m := w$ does not need to be blocked (because w stores information tagged M). Assignment $l := 1$ will execute.

Depending on whether $m := w$ is blocked, principals monitoring variable l (which is tagged L) either do or do not observe value 1 being assigned to l. The decision to block $m := w$ depends on the flow-sensitive label of w, which depends on sensitive information $m > 0$. So $m > 0$ is leaked if observers can detect that $m := w$ is blocked.[2]

To prevent such leaks, *metalabels* (e.g., [6]) might be introduced to represent the sensitivity of information encoded in flow-sensitive labels. For example, the metalabel for w in (2) would be M, corresponding to the sensitivity of information encoded in the flow-sensitive label tagging w. Only principals authorized to read information allowed by the metalabel (i.e., M) would be allowed to observe the label of w. The metalabel that tags w would also capture the sensitivity of the decision to execute $m := w$ and reach $l := 1$. To prevent the implicit flow of that information (which is tagged with M) to variable l (tagged L), assignment $l := 1$ must not be executed.

Since metalabels are flow-sensitive, they too could encode sensitive information that might leak to observers. It is tempting to employ meta-meta labels to prevent those leaks. However, flow-sensitive meta-meta labels might then leak. We seem to need a *label chain* associated with each variable: a label ℓ_1, metalabel ℓ_2, meta-meta label ℓ_3, etc.

This paper introduces and analyzes dynamic enforcement mechanisms that employ label chains of arbitrary length. We start by formalizing label chains (§2) and defining enforcers (§3). We next (§4) extend block-safe noninterference (BNI) [22] to stipulate that sensitive information does not leak to observers of variables and label chains. Enforcer ∞-*Enf* is derived (§5); it uses label chains of infinite length to enforce BNI. A family *k*-*Enf* of enforcers use finite label chains to approximate the infinite label chains of ∞-*Enf*. Our *k*-*Enf* enforcers also are shown (§6) to satisfy BNI.

There might be a *permissiveness* penalty when shorter label chains (e.g., finite label chains) are used to approximate longer ones (e.g., infinite label

[2]In fact, an arbitrary number of bits can be leaked through blocking executions [2].

chains). This paper formally characterizes the relationship between permissiveness and storage overhead of label chains having different lengths. We present theorems that relate label chain length and permissiveness for *k-Enf* enforcers (§7) as well as for other enforcers (§8) that satisfy BNI. The relationships between permissiveness and storage overhead depend on initialization, threat model, size of the lattice, as well as, on certain semantic attributes of enforcement mechanisms: *k-precise*, *k-varying*, and *k-dependent*.

2 Label Chains

Each variable x in a program will be associated with a possibly infinite label chain $\langle \ell_1, \ell_2, \ldots, \ell_i, \ell_{i+1}, \ldots \rangle$, where label ℓ_1 specifies sensitivity for the value stored in x and label ℓ_{i+1} specifies sensitivity for ℓ_i. Labels come from a possibly infinite *underlying* lattice $\mathcal{L} = \langle L, \sqsubseteq, \sqcup \rangle$ with bottom element $\perp$. For[3] $\ell, \ell' \in \mathcal{L}$, if $\ell \sqsubseteq \ell'$ holds, then ℓ' is *at least as restrictive as* ℓ, signifying that information is allowed to flow from data tagged with ℓ to data tagged with ℓ'.

Every principal p is assigned a fixed label ℓ that signifies p can read variables and labels whose sensitivity is at most ℓ. Thus, if variable x is tagged with ℓ' and p is assigned label ℓ, then p is allowed to read x iff $\ell' \sqsubseteq \ell$ holds.

Unless a label chain $\langle \ldots, \ell_i, \ell_{i+1}, \ldots \rangle$ is *monotonically decreasing*—$\ell_{i+1} \sqsubseteq \ell_i$ for $i \geq 1$— then sensitive information can be leaked. Here is why. Consider a variable x having non-monotonically decreasing label chain $\langle \mathsf{L}, \mathsf{H}, \ldots \rangle$, where $\mathsf{L} \sqsubset \mathsf{H}$. Principals assigned label L are authorized to read the value in x. When read access to x succeeds, these principals conclude that the label of x is L. Thus, success in reading x leaks to a principal assigned L information about the label of x—even though label chain $\langle \mathsf{L}, \mathsf{H}, \ldots \rangle$ defines the sensitivity of that label to be H. Such leaks cannot occur in monotonically decreasing label chains.

Label chains are implemented by sequencing individual labels stored in a memory M. Domain $dom(M)$ of a memory M includes:

- *Variables* that store (say) integers ($\nu \in \mathbb{Z}$). Lower case letters (e.g., a, w, x, h, m, l) denote variables. $M(x)$ is the integer stored in variable

[3]When $\mathcal{L} = \langle L, \sqsubseteq, \sqcup \rangle$, we write $\ell \in \mathcal{L}$ to assert that $\ell \in L$ holds.

x by M. Let *Var* be the set of variables. *Constants* (e.g., 1, 2, 3) are a subset of *Var* whose values are fixed.

- *Tags* that store labels ($\ell \in \mathcal{L}$) representing sensitivity. The label for x is stored at tag $T(x)$ in M; its value is $M(T(x))$. Some tags store labels representing the sensitivity of other tags. The label for $T^i(x)$ is stored in tag $T(T^i(x))$, for $i \geq 1$. We say *value* v when referring to either a label or an integer.
- *Auxiliaries* that store additional information needed by an enforcement mechanism (e.g., a stack to track implicit flows in nested **if** commands). The names of auxiliaries are μ_1, μ_2, etc.

Tags and auxiliaries are called *metadata*. A possibly infinite label chain $\langle T(q), T^2(q), \ldots, T^i(q), \ldots \rangle$ will be associated with each *identifier* q that is either a variable or a tag (but not an auxiliary). For convenience, we define $T^0(q) \triangleq q$ and $T^{i+1}(q) \triangleq T(T^i(q))$. We also may write $T^i(q)$ instead of $M(T^i(q))$ for that value in memory M if there will be no ambiguity (e.g., $T^i(q) \sqsubseteq \ell$, $T^i(q) \sqcup T^j(q')$). We require:

$$\forall i \geq 1\colon\ \ T^i(q) \in dom(M) \ \Rightarrow\ T^{i-1}(q) \in dom(M).$$

The mappings defined by M and T^i extend from identifiers to expressions (of variables or tags) $e \oplus e'$ in the usual way:

$$M(e \oplus e') \triangleq M(e) \oplus M(e') \tag{3}$$

$$T^i(e \oplus e') \triangleq T^i(e) \sqcup T^i(e'), \text{ for } i \geq 1. \tag{4}$$

Variables are categorized according to whether their label chains may change during execution. For a *flexible* variable w, the entire associated label chain might be updated when a value is assigned to w. So, the label chain of flexible variable w is flow-sensitive. For an *anchor* variable a, which can model a source or a sink of information with specific sensitivity, the label stored in $T(a)$ remains fixed throughout execution, and the remaining elements of the label chain satisfy:

$$M(T^i(a)) = \bot \text{ for any } T^i(a) \in dom(M) \text{ with } i > 1. \tag{5}$$

This form of chain is sensible for an anchor variable because $T(a)$ is declared in the program text and thus that label can be considered public (i.e., $T^2(a)$

is $\perp$) when execution starts. No other information can be encoded in $T(a)$ during execution because $T(a)$ remains fixed. So, $T(a)$ ought to be considered public during execution, too. The requirement that label chains be monotonically decreasing then leads to (5). A constant ν is a special case of an anchor variable:

$$M(T^i(\nu)) = \perp, \text{ for any } T^i(\nu) \in dom(M) \text{ with } i \geq 1. \tag{6}$$

3 Enforcers

Execution of a command C on a memory M can be represented by a *trace* τ, which is a potentially infinite sequence

$$\langle C_1, M_1 \rangle \rightarrow \langle C_2, M_2 \rangle \rightarrow \ldots \rightarrow \langle C_n, M_n \rangle \rightarrow \ldots$$

with $C_1 = C$. A *state* $\langle C_i, M_i \rangle$ gives the command C_i that will next be executed and gives a memory M_i to be used in that execution. A sequence τ' of states is considered a *subtrace* of τ iff $\tau = \ldots \rightarrow \tau' \rightarrow \ldots$. We write $|\tau|$ to denote the length of τ and $\tau[i]$ to denote the ith state in τ for $1 \leq i \leq |\tau|$. We also write $\langle C, M \rangle =^0 \langle C', M' \rangle$ to denote that two states agree on the command and the values in variables:

- $C = C'$,
- $dom(M) \cap Var = dom(M') \cap Var$, and
- $\forall x \in dom(M) \cap Var$: $M(x) = M'(x)$.

A set of operational semantics rules is employed to formally define traces. This paper uses a **while**-language (Figure 1) with operational semantics rules R (Figure 2). Notice, R does not reference metadata. Notation $M[x \mapsto \nu]$ in (AsgnA) and (AsgnF) defines a memory that equals M except x is mapped to ν. *Conditional delimiter* **exit** in rules for (If1), (If2), (Wl1), and (Wl2) marks the end of *conditional commands* (similar to[30, 32]). When execution of the corresponding taken branch completes, rule (Exit) is triggered.[4] Notice that C_i in a state $\langle C_i, M_i \rangle$ can be a command C as defined in Figure 1, a termination delimiter such as **stop**, or a command involving a conditional delimiter **exit**.

[4]For a **while** command, the number of times (Exit) is triggered equals the number of times rules (Wl1) and (Wl2) are invoked for this command.

$$
\begin{array}{lrcl}
\textit{(Constants)} & \nu & \in & \mathbb{Z} \\
\textit{(Anchor variables)} & a, x & \in & Var_A \\
\textit{(Flexible variables)} & w, x & \in & Var_F \\
\textit{(Expressions)} & e & ::= & \nu \mid x \mid e_1 \ \oplus \ e_2 \\
\textit{(Commands)} & C & ::= & \textbf{skip} \mid x := e \mid C_1; C_2 \mid \\
 & & & \textbf{if } e \textbf{ then } C_1 \textbf{ else } C_2 \textbf{ end} \mid \\
 & & & \textbf{while } e \textbf{ do } C \textbf{ end}
\end{array}
$$

Figure 1: Syntax

$$
(\textsc{Skip})\ \frac{}{\langle \textbf{skip}, M\rangle \rightarrow \langle \textbf{stop}, M\rangle}
\qquad
(\textsc{AsgnA})\ \frac{\nu = M(e)}{\langle a := e, M\rangle \rightarrow \langle \textbf{stop}, M[a \mapsto \nu]\rangle}
$$

$$
(\textsc{AsgnF})\ \frac{\nu = M(e)}{\langle w := e, M\rangle \rightarrow \langle \textbf{stop}, M[w \mapsto \nu]\rangle}
$$

$$
(\textsc{If1})\ \frac{M(e) \neq 0}{\langle \textbf{if } e \textbf{ then } C_1 \textbf{ else } C_2 \textbf{ end}, M\rangle \rightarrow \langle C_1; \textbf{exit}, M\rangle}
$$

$$
(\textsc{If2})\ \frac{M(e) = 0}{\langle \textbf{if } e \textbf{ then } C_1 \textbf{ else } C_2 \textbf{ end}, M\rangle \rightarrow \langle C_2; \textbf{exit}, M\rangle}
$$

$$
(\textsc{Wl1})\ \frac{M(e) \neq 0}{\langle \textbf{while } e \textbf{ do } C \textbf{ end}, M\rangle \rightarrow \quad \langle C; \textbf{while } e \textbf{ do } C \textbf{ end}; \textbf{exit}, M\rangle}
$$

$$
(\textsc{Wl2})\ \frac{M(e) = 0}{\langle \textbf{while } e \textbf{ do } C \textbf{ end}, M\rangle \rightarrow \langle \textbf{exit}, M\rangle}
$$

$$
(\textsc{Exit})\ \frac{}{\langle \textbf{exit}, M\rangle \rightarrow \langle \textbf{stop}, M\rangle}
\qquad
(\textsc{Seq1})\ \frac{\langle C_1, M\rangle \rightarrow \langle \textbf{stop}, M'\rangle}{\langle C_1; C_2, M\rangle \rightarrow \langle C_2, M'\rangle}
$$

$$
(\textsc{Seq2})\ \frac{\langle C_1, M\rangle \rightarrow \langle C_1', M'\rangle \qquad C_1' \neq \textbf{stop}}{\langle C_1; C_2, M\rangle \rightarrow \langle C_1'; C_2, M'\rangle}
$$

Figure 2: Structural operational semantics R

The rules comprising R define a function $trace_R(C, M)$ that maps a command C and a memory M to the trace that represents the entire execution of C started with initial memory M. For $trace_R(C, M)$ to be well-defined, M should be *healthy for* C denoted $M \models \mathcal{H}(C)$ and formalized below, where $x{\in}C$ indicates that $x{\in} Var$ appears in C:

$$\begin{aligned} M \models \mathcal{H}(C) \;\; &\triangleq \;\; \forall x{\in} Var\colon (x \in C \Rightarrow x \in dom(M)) \\ &\qquad\qquad\qquad \wedge (x \in dom(M) \Rightarrow M(x) \in \mathbb{Z}). \end{aligned}$$

By definition, If $trace_R(C, M)$ is finite, then it ends with *normal termination* state $\langle \mathbf{stop}, M' \rangle$.

Executing command C on memory M under the auspices of an *enforcer* E leads to a trace $\tau = trace_E(C, M)$. We expect that such traces will satisfy some policy P of interest, where E blocks traces if needed to satisfy P. So, a trace $\tau = trace_E(C, M)$ may end with *blocked state* $\langle \mathbf{block}, M' \rangle$; omitting the blocked state from τ and projecting only commands and variables should yield a prefix of $trace_R(C, M)$.

We now formalize the definition of an enforcer. Define $blk(\tau)$ to hold iff τ ends with a blocked state, and define prefix relation $\tau \preceq \tau'$ on traces:

$$\begin{aligned} \tau \preceq \tau' \;\; \triangleq \;\; & |\tau| \le |\tau'| \;\; \wedge \;\; l = |\tau| \;\; \wedge \;\; (\forall 1 {\le} i {<} l\colon \tau[i] =^0 \tau'[i]) \\ & \wedge \;\; (\neg blk(\tau) \Rightarrow \tau[l] =^0 \tau'[l])) \end{aligned}$$

E is an enforcer on R for P if

- $(\forall C, M\colon trace_E(C, M) \preceq trace_R(C, M))$ and
- the image of function $trace_E$, which is a set of traces, satisfies P.

An enforcer E might employ metadata, including label chains of size $n_E \ge 1$ and a set Aux_E of auxiliaries.

A memory M is *healthy* for enforcer E, lattice $\mathcal{L}$, and command C denoted by $M \models \mathcal{H}(E, \mathcal{L}, C)$, iff

- $M \models \mathcal{H}(C)$,
- for each variable x, $dom(M)$ includes exactly n_E tags comprising a label chain, where all tags are mapped to lattice $\mathcal{L}$:

$$\begin{aligned} &\forall x \in dom(M) \cap Var\colon \\ &\qquad (\forall 1 \le i \le n_E\colon T^i(x) \in dom(M) \wedge M(T^i(x)) \in \mathcal{L}) \\ &\qquad \wedge (\forall i > n_E\colon T^i(x) \notin dom(M)) \end{aligned}$$

– $dom(M)$ contains requisite auxiliaries Aux_E:

$$\forall \mu \in Aux_E\colon \mu \in dom(M)$$

– flexible variables in M have monotonically decreasing label chains,

– anchor variables in M have label chains satisfying (5), and

– constants in M have label chains satisfying (6).

Notice, if $M \models \mathcal{H}(E, \mathcal{L}, C)$ and $x \in dom(M)$, then sensitivity $T^{n_E+1}(x)$ of last element $T^{n_E}(x)$ does not belong to $dom(M)$, and thus, $T^{n_E+1}(x)$ is not defined.

For mapping $Init_E$ from auxiliaries in Aux_E to initial values, a memory M is defined to be *initially healthy* for enforcer E, lattice $\mathcal{L}$, and command C denoted $M \models \mathcal{H}_0(E, \mathcal{L}, C)$ iff:

– $M \models \mathcal{H}(E, \mathcal{L}, C)$, and

– auxiliaries Aux_E are initialized according to $Init_E$:

$$\forall \mu \in Aux_E\colon M(\mu) = Init_E(\mu).$$

We consider enforcers $E \triangleq \langle trace_E, n_E, Aux_E, Init_E \rangle$ satisfying:

(E1) Trace $trace_E(C, M)$ is defined when $M \models \mathcal{H}_0(E, \mathcal{L}, C)$ holds.

(E2) For a memory M_i in any state of $trace_E(C, M)$, condition $M_i \models \mathcal{H}(E, \mathcal{L}, C)$ holds.

(E3) E updates the label chain of a flexible variable w only in performing an assignment to w, or at **exit** for a conditional command whose branches (taken or untaken) contain an assignment to w.[5]

[5]More formally, if $\langle C_1; C_2, M_1 \rangle \xrightarrow{*} \langle C_1'; C_2, M_2 \rangle$ is a subtrace of $trace_E(C, M)$, where C_1 is a subcommand of C, and if "$w := e$"$\notin C_1$ holds for a flexible variable w, then the following should hold:

$$(\forall i \geq 1\colon T^i(w) \in dom(M_1) \Rightarrow M_1(T^i(w)) = M_2(T^i(w)))$$

Note, a conditional delimiter (e.g., **exit**) is not considered a subcommand of C.

4 Threat Models and BNI

Observations Our threat model has principals observing updates to identifiers. When an assignment to a flexible variable w is executed, each element in set $O(w) \triangleq \{w, T(w), \ldots, T^i(w), \ldots\}$ is updated. When an assignment to an anchor variable a is executed, only $O(a) \triangleq \{a\}$ is updated. A principal p assigned label ℓ observes updates to variables and tags q, where $T(q)$ is in the domain of a memory M and $M(T(q)) \sqsubseteq \ell$ holds. A similar threat model is used in [5].

Principals do not observe updates to an identifier q when $T(q) \notin \mathit{dom}(M)$ holds, because q then is not covered by the security policy to be enforced. That implies principals do not observe updates to the last element of a label chain. Also, a principal p assigned ℓ might be allowed to observe updates to an identifier $T^j(q)$ (i.e., $T^{j+1}(q) \sqsubseteq \ell$) but p might not be allowed to observe updates to a preceding identifier $T^i(q)$ (i.e., $T^{i+1}(q) \not\sqsubseteq \ell$) for $0 \leq i < j$, due to monotonically decreasing label chains.

We now formalize the *observation* available to a principal p assigned label ℓ when an assignment executes. Define the projection $M|_\ell^S$ of a memory M with respect to label ℓ and a set S of identifiers:

$$M|_\ell^S \triangleq \{\langle q, M(q)\rangle \mid q{\in}S \ \wedge\ T(q){\in}\mathit{dom}(M) \ \wedge\ M(T(q)){\sqsubseteq}\ell\}$$

If an assignment to a variable x is performed and memory M results, then observation $M|_\ell^{O(x)}$ is generated to p. Notice, $M|_\ell^{O(x)}$ can be empty.

A sequence of observations is generated along with a trace. Given a trace τ, define $\tau|_\ell^S$ to be a sequence $\theta = \Theta_1 \rightharpoonup \ldots \rightharpoonup \Theta_n$ of those observations involving identifiers in set S and having sensitivity at most ℓ:

$$\epsilon|_\ell^S \triangleq \epsilon \qquad\qquad \langle C, M\rangle|_\ell^S \triangleq \epsilon$$

$$(\langle C, M\rangle \to \langle C', M'\rangle \to \tau)|_\ell^S \triangleq$$
$$\begin{cases} M'|_\ell^{O(x)\cap S} \rightharpoonup (\langle C', M'\rangle \to \tau)|_\ell^S, & \text{if } C \text{ is ``}x := e; C'\text{''} \\ (\langle C', M'\rangle \to \tau)|_\ell^S, & \text{otherwise} \end{cases}$$

When $S = \{T^i(x) \mid 0 \leq i \leq k \ \wedge\ x \in \mathit{Var}\}$, we abbreviate $\tau|_\ell^S$ by $\tau|_\ell^k$. We write $\theta =_{obs} \theta'$ to specify equality of sequences of observations with empty observations omitted, since θ and θ' are then equivalent for principals.

This *strong threat model* produces observations for both variables and tags. But observations are not generated when identifiers are updated at execution points other than assignments to variables (e.g., no observation is generated when a conditional delimiter **exit** is executed). This is because enforcers may differ about whether these other updates are performed. And those differences would make comparison of observations (employed in §7) problematic.

Block-safe Noninterference Block-safe Noninterference (BNI) is a form of *noninterference* [16] that incorporates observations on tags and considers all finite traces—normally terminated and blocked. Formally, BNI stipulates that if two finite traces of the same command agree on initial values whose sensitivity is at most ℓ, then observations (involving variables and tags) visible to a principal assigned label ℓ should be the same. We define for $k \geq 0$ specialized k-BNI that restricts observations to variables and tags $T^0, T^1, \ldots, T^k$; and $M|_\ell$ abbreviates $M|_\ell^{dom(M)}$.

$$
\begin{aligned}
k\text{-BNI}(E, \mathcal{L}, C) \;\triangleq\; (\forall \ell \in \mathcal{L}\colon \forall M, M'\colon &\\
& M \models \mathcal{H}_0(E, \mathcal{L}, C)\\
& \wedge M' \models \mathcal{H}_0(E, \mathcal{L}, C)\\
& \wedge M|_\ell = M'|_\ell\\
& \wedge \tau = \mathit{trace}_E(C, M) \text{ is finite}\\
& \wedge \tau' = \mathit{trace}_E(C, M') \text{ is finite}\\
& \Rightarrow \tau|_\ell^k =_{obs} \tau'|_\ell^k)
\end{aligned}
$$

If k-BNI$(E, \mathcal{L}, C)$ holds for every C, then E enforces k-BNI$(\mathcal{L})$.[6] If for all $k \geq 0$ and $\mathcal{L}$, enforcer E satisfies k-BNI$(\mathcal{L})$, then we say that E enforces BNI.

0-BNI is stronger than *termination insensitive noninterference* (TINI) [37] enforced by [3, 10, 12, 14, 25]. TINI concerns normally terminated executions but does not consider finite traces that correspond to blocked executions. So TINI ignores traces that become blocked by the enforcement mechanism and thereby leak sensitive information. 0-BNI considers all finite traces. So, an enforcement mechanism that satisfies 0-BNI will satisfy TINI, too.

0-BNI is weaker than *termination sensitive noninterference* (TSNI) [36]. TSNI considers infinite and finite traces (terminated normally as well as

[6]Notice that if E satisfies $(k+1)$-BNI$(\mathcal{L})$, then E satisfies k-BNI$(\mathcal{L})$.

blocked). Because 0-BNI ignores infinite traces, 0-BNI allows leaks through termination channels that already exist in a program (due to non-terminating while-loops).

We chose to study 0-BNI, so we could focus on leaks introduced by the enforcer itself. The enforcement techniques of the next section prevent those leaks. Moreover, they can be extended to enforce TSNI (e.g., would leak through non-terminating while-loops) using techniques similar to those given in [6].

5 Enforcer ∞-*Enf*

We use familiar insights about information flow to formulate an enforcer ∞-*Enf* that uses infinite label chains (i.e., $n_{\infty\text{-}Enf} = \infty$) to enforce BNI for programs written in the programming language of Figure 1. We later derive from ∞-*Enf* the k-*Enf* family of enforcers that use finite label chains.

5.1 Updating Label Chains of Flexible Variables

When assignment $w := e$ executes in isolation, the value of e flows explicitly to flexible variable w. So, w should be at least as sensitive as e. Therefore, just prior to the assignment, ∞-*Enf* updates tag $T(w)$ with $T(e)$. But with that update, the value of $T(e)$ flows explicitly to $T(w)$, so ∞-*Enf* also must update tag $T^2(w)$ with $T^2(e)$. Repeating the argument, we conclude that when executing $w := e$, enforcer ∞-*Enf* should update tag $T^i(w)$ with $T^i(e)$, for $i \geq 0$.

Information can also flow implicitly from the *context* of an assignment to the target variable of that assignment. Context *ctx* of a command C is a set of boolean expressions that includes all guards involved in determining that C should be reached. If C appears in the body of a conditional command having guard e, then e belongs to the context of C. For example, consider:

$$\textbf{if } x > 0 \textbf{ then } w := w' \textbf{ else } w := w'' \textbf{ end} \tag{7}$$

Here, context *ctx* of $w := w'$ and $w := w''$ is $\{x > 0\}$. Notice, if $T^i(w') \neq T^i(w'')$ holds prior to (7) for some $i \geq 0$, then the value in $T^i(w)$ after the **if** command depends on *ctx*. Context *ctx* is prevented from leaking through $T^i(w)$ if we require that $T(ctx) \sqsubseteq T^{i+1}(w)$ holds, where $T(ctx)$ is the sensitivity of *ctx*.

In general, for q a flexible variable or a tag, if q is assigned the value of e (for e an expression of variables or tags), then information can flow explicitly from e to q and implicitly from ctx to q. Thus, sensitivity $T(q)$ of q should be updated to $T(e) \sqcup T(ctx)$. But, this update might also require updating $T^i(q)$ for $i \geq 1$. $UT(q, e, ctx)$ below describes tag updates triggered by q being updated with e in context ctx:

$$\begin{aligned} UT(q, e, ctx) \triangleq\ & T(q) := T(e) \sqcup T(ctx); \\ & UT(T(q), T(e) \sqcup T(ctx), ctx) \end{aligned}$$

For $w := e$ in context ctx, $UT(w, e, ctx)$ expands to[7]

$$\forall i \geq 1\colon\ T^i(w) := T^i(e) \sqcup T(ctx). \tag{8}$$

So, enforcer ∞-*Enf* will produce a new label chain for w; each label in that chain is computed according to (8).

5.2 Preventing Leaks through Anchor Variables

Prior to executing $a := e$ for an anchor variable a, an enforcer checks a *block condition* $G_{a:=e}$. If $G_{a:=e}$ holds, then the explicit and implicit flows to a in $a := e$ do not constitute leaks; if $G_{a:=e}$ does not hold, then execution blocks.

But blocking execution might cause implicit flow of sensitive information, as seen with (2). We avoid this flow by generalizing the definition of ctx to include block conditions that could have already been checked. This generalization is consistent with the role of ctx: execution of $a := e$ and of any command that might follow is conditioned on whether $G_{a:=e}$ holds. If execution of C depends on $G_{a:=e}$ being *true*, then $G_{a:=e}$ belongs to the context ctx of C.

We now show how to construct $G_{a:=e}$ for an assignment $a := e$ in context ctx. The value of e explicitly flows to a. So, a should be at least as sensitive as e: $T(e) \sqsubseteq T(a)$. Because execution of $a := e$ depends on $G_{a:=e}$, the context of $a := e$ is $ctx \cup \{G_{a:=e}\}$. Information flows implicitly from this context to a. Variable a should thus be at least as sensitive as $T(ctx \cup \{G_{a:=e}\})$. We thus require $T(ctx) \sqcup T(G_{a:=e}) \sqsubseteq T(a)$. So, for $G_{a:=e}$ to hold, both $T(e) \sqsubseteq T(a)$ and $T(ctx) \sqcup T(G_{a:=e}) \sqsubseteq T(a)$ should hold. We conclude

$$G_{a:=e}\ \Rightarrow\ (T(e) \sqsubseteq T(a)\ \wedge\ T(ctx) \sqcup T(G_{a:=e}) \sqsubseteq T(a))$$

[7] This expansion uses the fact that the label chain associated with ctx is monotonically decreasing: $T^{i+1}(ctx) \sqsubseteq T^i(ctx)$.

or equivalently

$$G_{a:=e} \;\Rightarrow\; (T(e) \sqcup T(ctx) \sqcup T(G_{a:=e}) \sqsubseteq T(a))\,. \tag{9}$$

One possible solution for $G_{a:=e}$ in (9) is:

$$G_{a:=e} \triangleq (T(e) \sqcup T(ctx) \sqsubseteq T(a)). \tag{10}$$

To verify that (10) is a solution, first compute sensitivity:

$$\begin{aligned} T(G_{a:=e}) &= T(\; T(e) \sqcup T(ctx) \sqsubseteq T(a) \;) \\ &= T^2(e) \sqcup T^2(ctx) \sqcup T^2(a) && \{\text{due to (4)}\} \\ &= T^2(e) \sqcup T^2(ctx) \sqcup \bot && \{T^2(a) = \bot\} \\ &= T^2(e) \sqcup T^2(ctx) && \{\ell \sqcup \bot = \ell\} \end{aligned} \tag{11}$$

Substituting $T^2(e) \sqcup T^2(ctx)$ for $T(G_{a:=e})$, substituting $T(e) \sqcup T(ctx) \sqsubseteq T(a)$ for $G_{a:=e}$ in (9), and noticing that $T^2(e) \sqcup T^2(ctx) \sqsubseteq T(e) \sqcup T(ctx)$ (due to monotonically decreasing label chains), equation (9) becomes equivalent to a true statement, which is what we needed to verify solution (10).

$G_{a:=e}$ in (10) is used by all dynamic flow sensitive enforcement mechanisms we know. But, we seem to be the first to present it as a solution of (9).

5.3 Operational Semantics for ∞-*Enf*

Enforcer ∞-*Enf* uses (i) UT (see (8)) for deducing label chains and (ii) $G_{a:=e}$ (see (10)) for blocking possibly unsafe assignments. UT and $G_{a:=e}$ mention tags for variables and sensitivity $T(ctx)$ of the context but do not need ctx, $T^2(ctx)$, $T^3(ctx)$, etc. $T(ctx)$ is the join of the sensitivity of each guard and each block condition that determines the reachability of a command. ∞-*Enf* uses auxiliaries to maintain $T(ctx)$:

- *cc* (conditional context) keeps track of the sensitivity of the guards in all conditional commands that encapsulate the next command to be executed, and

- *bc* (blocking context) keeps track of the sensitivity of information revealed by block conditions that might influence reachability of the next command executed.

So, $Aux_{\infty\text{-}Enf} = \{cc, bc\}$. We now show how $T(ctx)$ is defined in terms of cc and bc.

Auxiliary bc is a tag that (conservatively) stores a label at least as restrictive as the sensitivity of all block conditions that could have already been evaluated. Any observation after assignment $a := e$ reveals information about $G_{a:=e}$ and about context ctx in which $G_{a:=e}$ is evaluated. So, whenever a block condition $G_{a:=e}$ is checked, ∞-*Enf* updates bc with $T(G_{a:=e})$ and $T(ctx)$:

$$bc := T(G_{a:=e}) \sqcup T(ctx). \tag{12}$$

From (11) and monotonicity of label chains (i.e., $T^2(ctx) \sqsubseteq T(ctx)$), we then get:

$$bc := T^2(e) \sqcup T(ctx). \tag{13}$$

No block condition has been evaluated before execution starts, so bc is initialized to $\bot$: $Init_{\infty\text{-}Enf}(bc) = \bot$.

Auxiliary cc is implemented in ∞-*Enf* using a stack. Whenever execution enters a conditional command, the sensitivity of the corresponding guard is pushed onto cc; upon exit the top element of cc is popped. $\lfloor cc \rfloor$ will denote the join of all labels in cc. At the beginning of execution, no conditional command has been entered, so cc is initialized to the empty stack ϵ with $\lfloor \epsilon \rfloor \triangleq \bot$. So, we have $Init_{\infty\text{-}Enf}(cc) = \epsilon$.

Putting all together, sensitivity $T(ctx)$ is $\lfloor cc \rfloor \sqcup bc$. Substituting $\lfloor cc \rfloor \sqcup bc$ for $T(ctx)$ in (10), block condition $G_{a:=e}$ becomes:

$$T(e) \sqcup \lfloor cc \rfloor \sqcup bc \sqsubseteq T(a). \tag{14}$$

Substituting $\lfloor cc \rfloor \sqcup bc$ for $T(ctx)$ in (13), the update of bc becomes:

$$bc := T^2(e) \sqcup \lfloor cc \rfloor \sqcup bc. \tag{15}$$

So, $G_{a:=e}$ and the update of bc have now been expressed in terms of tags and auxiliaries that ∞-*Enf* uses.

Rule (AsgnA) in Figure 3 uses (14) and (15). If $G_{a:=e}$ does not hold, then rule (AsgnAFail) is triggered. Notice that in (AsgnAFail), bc is updated with a label representing the sensitivity of the context in which execution is blocked. That label in bc dictates which principals are allowed to learn why an execution ended (i.e., due to a **block** versus due to a **stop**) without sensitive information leaking.

$$(\textsc{Skip})\ \frac{}{\langle \mathbf{skip}, M\rangle \rightarrow \langle \mathbf{stop}, M\rangle}$$

$$(\textsc{AsgnA})\ \frac{v = M(e) \qquad G_{a:=e} \qquad \ell = M(T^2(e)) \sqcup M(\lfloor cc \rfloor) \sqcup M(bc)}{\langle a := e, M\rangle \rightarrow \langle \mathbf{stop}, M[a \mapsto v, bc \mapsto \ell]\rangle}$$

$$(\textsc{AsgnAFail})\ \frac{v = M(e) \qquad \neg G_{a:=e} \qquad \ell = M(T^2(e)) \sqcup M(\lfloor cc \rfloor) \sqcup M(bc)}{\langle a := e, M\rangle \rightarrow \langle \mathbf{block}, M[bc \mapsto \ell]\rangle}$$

$$(\textsc{AsgnF})\ \frac{v_0 = M(e) \qquad \forall i \geq 1\colon v_i = M(T^i(e)) \sqcup M(\lfloor cc \rfloor) \sqcup M(bc)}{\langle w := e, M\rangle \rightarrow \langle \mathbf{stop}, M[\forall i \geq 0 \colon T^i(w) \mapsto v_i]\rangle}$$

$$G_{a:=e} \text{ is } M(T(e)) \sqcup M(\lfloor cc \rfloor) \sqcup M(bc) \sqsubseteq M(T(a))$$

Figure 3: Operational semantics for **skip** and assignments.

Rule ($\textsc{AsgnF}$) for assignment $w := e$ to flexible variable w implements (8), given $T(ctx) = \lfloor cc \rfloor \sqcup bc$. So, the label chain of w is updated as follows:

$$\forall i \geq 1\colon T^i(w) := T^i(e) \sqcup \lfloor cc \rfloor \sqcup bc.$$

Rules for conditional commands are given in Figure 4. They adopt techniques employed by other dynamic enforcement mechanisms (e.g., [14]) to update auxiliary cc and handle implicit flows to variables and metadata that could have been updated in untaken branches. When execution reaches a conditional command C, tuple $\langle \ell, W, A\rangle$ is pushed onto cc (writing $M(cc).push(\langle \ell, W, A\rangle))$; when execution exits C, tuple $\langle \ell, W, A\rangle$ is popped. Here we define the elements of tuple $\langle \ell, W, A\rangle$.

- Element ℓ is the sensitivity of the guard e of conditional command C. Including ℓ in cc while taken branch C_t of C is executed signifies that the sensitivity of the context of C_t is the result of augmenting the sensitivity of the context of C with the sensitivity of guard e.

- Element W is set $targetFlex(C_u)$ of target flexible variables in untaken branch C_u of C. If $w \in W$, then $T^i(w)$ for $i \geq 0$ could have been updated if C_u were executed. To capture implicit flow from the context of C_u to $T^i(w)$, when execution exits C, sensitivity $T^{i+1}(w)$ is augmented with the sensitivity of the context of C_u, which is the same as the context of C_t.

$$(\textsc{If1})\ \frac{\begin{array}{c} M(e) \neq 0 \qquad W = \mathit{targetFlex}(C_2) \\ A = \mathit{targetAnchor}(C_2) \qquad cc' = M(cc).\mathit{push}(\langle M(T(e)),\ W,\ A\rangle) \end{array}}{\langle \textbf{if}\ e\ \textbf{then}\ C_1\ \textbf{else}\ C_2\ \textbf{end}, M\rangle \to \langle C_1; \textbf{exit}, M[cc \mapsto cc']\rangle}$$

$$(\textsc{If2})\ \frac{\begin{array}{c} M(e) = 0 \qquad W = \mathit{targetFlex}(C_1) \\ A = \mathit{targetAnchor}(C_1) \qquad cc' = M(cc).\mathit{push}(\langle M(T(e)),\ W,\ A\rangle) \end{array}}{\langle \textbf{if}\ e\ \textbf{then}\ C_1\ \textbf{else}\ C_2\ \textbf{end}, M\rangle \to \langle C_2; \textbf{exit}, M[cc \mapsto cc']\rangle}$$

$$(\textsc{Wl1})\ \frac{M(e) \neq 0 \qquad cc' = M(cc).\mathit{push}(\langle M(T(e)),\ \emptyset,\ \emptyset\rangle)}{\langle \textbf{while}\ e\ \textbf{do}\ C\ \textbf{end}, M\rangle \to \qquad \langle C; \textbf{while}\ e\ \textbf{do}\ C\ \textbf{end}; \textbf{exit}, M[cc \mapsto cc']\rangle}$$

$$(\textsc{Wl2})\ \frac{\begin{array}{c} M(e) = 0 \qquad W = \mathit{targetFlex}(C) \\ A = \mathit{targetAnchor}(C) \qquad cc' = M(cc).\mathit{push}(\langle M(T(e)),\ W,\ A\rangle) \end{array}}{\langle \textbf{while}\ e\ \textbf{do}\ C\ \textbf{end}, M\rangle \to \langle \textbf{exit}, M[cc \mapsto cc']\rangle}$$

$$(\textsc{Exit})\ \frac{\begin{array}{c} bc' = \begin{cases} M(bc) \sqcup M(\lfloor cc \rfloor), & \text{if } M(cc).\mathit{top}.A \neq \emptyset \\ M(bc), & \text{otherwise} \end{cases} \\ M' = U(M, M(cc).\mathit{top}.W) \qquad cc' = cc.\mathit{pop} \end{array}}{\langle \textbf{exit}, M\rangle \to \langle \textbf{stop}, M'[cc \mapsto cc', bc \mapsto bc']\rangle}$$

$$\begin{array}{l} U(M, W) \triangleq \\ \quad M[\ \forall w{\in}W\colon \forall i \geq 1\colon T^i(w) \mapsto T^i(w) \sqcup M(\lfloor cc \rfloor) \sqcup M(bc)] \end{array}$$

Figure 4: Operational semantics for conditional commands.

– Element A is set $\mathit{targetAnchor}(C_u)$ of all anchor variables in untaken branch C_u. If A is not empty and if C_u would have been executed, then a block condition could have been evaluated, possibly causing that execution to be blocked. So, reachability of a command following C might be influenced by whether C_u has been executed, and thus, it might be influenced by the context of C_u. So, when execution exits C, auxiliary bc is augmented with the sensitivity of the context of C_u (which is the same as the context of C_t).

Figure 5 gives rules for executing sequences of commands. Rule (SeqF) asserts that execution stops once an assignment is blocked.

Given a lattice $\mathcal{L}$, a command C, and a memory M initially healthy for ∞-*Enf*, $\mathcal{L}$, and C, function $\mathit{trace}_{\infty\text{-}\mathit{Enf}}(C, M)$ is defined by the operational

$$(\textsc{Seq1})\ \frac{\langle C_1, M\rangle \to \langle \mathbf{stop}, M'\rangle}{\langle C_1; C_2, M\rangle \to \langle C_2, M'\rangle}$$

$$(\textsc{Seq2})\ \frac{\langle C_1, M\rangle \to \langle C_1', M'\rangle \qquad C_1' \notin \{\mathbf{stop}, \mathbf{block}\}}{\langle C_1; C_2, M\rangle \to \langle C_1'; C_2, M'\rangle}$$

$$(\textsc{SeqF})\ \frac{\langle C_1, M\rangle \to \langle \mathbf{block}, M'\rangle}{\langle C_1; C_2, M\rangle \to \langle \mathbf{block}, M'\rangle}$$

Figure 5: Operational semantics for sequences

semantics presented in Figures 3, 4, and 5. We prove the following Theorem in the Appendix.

Theorem 1. *∞-Enf is an enforcer on R for BNI.*

6 Enforcer *k-Enf*

An enforcer that uses infinite label chains cannot always be implemented with finite memory. But an infinite label chain can be approximated by a finite label chain. First notice that infinite label chain $\Omega = \langle \ell_1, \ldots, \ell_k, \ell_{k+1}, \ell_{k+2}, \ldots\rangle$ is *conservatively approximated* by infinite label chain $\Omega' = \langle \ell_1, \ldots, \ell_k, \ell_k, \ell_k, \ldots\rangle$, where k^{th} label ℓ_k is infinitely repeated. It is a conservative approximation, because if Ω' allows a principal p assigned label ℓ to observe the i^{th} element of Ω', then Ω allows p to observe the i^{th} element of Ω, too (but not *vice versa*). This is because Ω and Ω' agree up to the k^{th} element and, for $i \geq k$, the i^{th} element in Ω' is at least as restrictive as the corresponding element in Ω due to monotonically decreasing label chains: $\ell_{k+1} \sqsubseteq \ell_k$, $\ell_{k+2} \sqsubseteq \ell_k$, etc. Finite label chain with $m \geq 0$:

$$\Omega'' = \langle \ell_1, \ldots, \ell_k, \underbrace{\ell_k, \ldots, \ell_k}_{m}\rangle$$

also is a conservative approximation for Ω' (recall no observation is allowed for identifiers whose sensitivity is not defined). Consequently, an infinite label chain Ω can be approximated by finite label chain Ω''.

We employ such finite approximations to derive enforcer *k-Enf* from ∞-*Enf*. Enforcer *k-Enf* uses the operational semantics rules of ∞-*Enf* to

$$\text{(AsgnF)}\ \frac{v_0 = M(e) \qquad \forall 1 \leq i \leq k\colon v_i = M(T^i(e)) \sqcup M(\lfloor cc \rfloor) \sqcup M(bc)}{\langle w := e, M\rangle \to \langle \mathbf{stop}, M[\forall i\,{:}\,0 \leq i \leq k\,{:}\,T^i(w) \mapsto v_i]\rangle}$$

$$U(M, W) \triangleq$$
$$M[\,\forall w{\in}W\colon \forall i\,{:}\,1 \leq i \leq k\colon T^i(w) \mapsto T^i(w) \sqcup M(\lfloor cc \rfloor) \sqcup M(bc)].$$

Figure 6: Modified rules for *k-Enf*

compute up to the k^{th} tag. Because rule (AsgnA) mentions $T^2(x)$, we require $k \geq 2$. In ∞-*Enf*, only (AsgnF) and function U refers to $T^i(x)$ for $i > 2$. So in *k-Enf* rule (AsgnF) and function U are modified to compute labels only for the first k tags. See Figure 6.

Enforcer *k-Enf* generates observations for updates up to the k^{th} tag. To generate an observation about an update to the k^{th} tag, *k-Enf* conservatively approximates the sensitivity of element $T^k(x)$ to be itself. So, *k-Enf* actually is using label chains of length $n_{k\text{-}Enf} = k + 1$ and it conservatively approximates an infinite label chain $\Omega = \langle \ell_1, \ldots, \ell_k, \ell_{k+1}, \ell_{k+2}, \ldots\rangle$ that would have been computed by ∞-*Enf* with finite label chain $\Omega'' = \langle \ell_1, \ldots, \ell_k, \ell_k\rangle$.

Similar to ∞-*Enf*, enforcer *k-Enf* has $Aux_{k\text{-}Enf} = \{cc, bc\}$, $Init_{k\text{-}Enf}(cc) = \epsilon$, and $Init_{k\text{-}Enf}(bc) = \bot$. We prove the following theorem in the Appendix.

Theorem 2. *k-Enf is an enforcer on R for k-BNI($\mathcal{L}$), for any lattice $\mathcal{L}$ and $k \geq 2$.*

7 Permissiveness of *k-Enf* versus Chain Length

Approximation by shorter label chains has a penalty: permissiveness. The details however are not straightforward. For *k-Enf* enforcers, the penalty of shorter label chains will depend on the threat model and on assumptions about initialization. This section gives theorems to characterize that trade-off. And in the next section, we examine other classes of enforcers.

An enforcer E' is *at least as permissive as* an enforcer E if, for all executions of each command, E' emits observations involving at least as many identifiers as E. This comparison involves deciding whether identifiers (i.e., variables and tags) that appear in a sequence θ of observations produced by

E, also appear in a sequence θ' produced by E'. We define $\theta \trianglelefteq \theta'$ as follows

$$
\begin{aligned}
&\theta \trianglelefteq \theta' \triangleq \\
&\quad |\theta| \leq |\theta'| \ \wedge \ (\forall i \colon 1 \leq i \leq |\theta| \colon \mathit{dom}(\theta[i]) \subseteq \mathit{dom}(\theta'[i]))
\end{aligned}
$$

where $\theta[i]$ is the ith observation in sequence θ. Relation $\trianglelefteq$ does not depend on values being stored in variables because enforcers E and E' are required to compute the same values while executing the same command.

We compare permissiveness of enforcers relative to an underlying lattice and some identifiers of interest. We start the comparison with pairs of memories that satisfy some desired initialization condition, such as equality on initial values and label chains. We define an enforcer E' to be at least as permissive as an enforcer E for initialization condition ρ, underlying lattice $\mathcal{L}$, and identifiers up to the kth tag (i.e., T^k) with $k \geq 0$, writing $E \leq_{\rho}^{k,\mathcal{L}} E'$, as follows:

$$
\begin{aligned}
&E \leq_{\rho}^{k,\mathcal{L}} E' \ \triangleq \\
&\quad \forall C, M, M' \colon \rho(M, M') \\
&\qquad \wedge M \models \mathcal{H}_0(E, \mathcal{L}, C) \wedge M' \models \mathcal{H}_0(E', \mathcal{L}, C) \\
&\qquad \Rightarrow (\forall \ell \in \mathcal{L} \colon \mathit{trace}_E(C, M)|_{\ell}^{k} \trianglelefteq \mathit{trace}_{E'}(C, M')|_{\ell}^{k})
\end{aligned}
\tag{16}
$$

Notice, the consequent in definition (16) holds iff labels deduced by E are at least as restrictive as labels deduced by E'. Relation $\leq_{\rho}^{k,\mathcal{L}}$ is a preorder (i.e., reflexive and transitive relation) on enforcers.

For convenience, we introduce abbreviations:

– $E <_{\rho}^{k,\mathcal{L}} E' \ \triangleq \ E \leq_{\rho}^{k,\mathcal{L}} E' \wedge E' \not\leq_{\rho}^{k,\mathcal{L}} E$

– $E \cong_{\rho}^{k,\mathcal{L}} E' \ \triangleq \ E \leq_{\rho}^{k,\mathcal{L}} E' \wedge E' \leq_{\rho}^{k,\mathcal{L}} E$

Notice that from (16) we can prove that if $\rho \Rightarrow \rho'$, then

$$E \leq_{\rho'}^{k,\mathcal{L}} E' \Rightarrow E \leq_{\rho}^{k,\mathcal{L}} E'. \tag{17}$$

Also, if $k \leq k'$, then $E \leq_{\rho}^{k',\mathcal{L}} E' \Rightarrow E \leq_{\rho}^{k,\mathcal{L}} E'$.

We now examine how lengths of label chains relate to the permissiveness of enforcers by comparing the permissiveness of enforcers k-*Enf* and $(k+1)$-*Enf* for $k \geq 2$. To perform this comparison, the initial memories considered by k-*Enf* and $(k+1)$-*Enf* for executing a command should agree on values in variables and on labels in tags, up to the kth. Define

$$\begin{aligned} M|^k \triangleq \{\langle T^i(x), M(T^i(x))\rangle \mid 0 \leq i \leq k \\ \wedge\ x \in \mathit{Var}\ \wedge\ T^i(x) \in \mathit{dom}(M)\} \end{aligned}$$

The desired initialization condition then is:

$$\rho_k(M, M') \quad \triangleq \quad M|^k = M'|^k$$

Thus, initialization condition ρ_k allows a flexible variable w to be initially associated with label chains, where $\ell_{k+1} \sqsubset \ell_k$:

- $\Omega = \langle \ell_1, \ell_2, \ldots, \ell_{k-1}, \ell_k, \ell_{k+1}, \ell_{k+1}\rangle$ by $(k+1)$-*Enf*,
- $\Omega' = \langle \ell_1, \ell_2, \ldots, \ell_{k-1}, \ell_k, \ell_k\rangle$ by k-*Enf*.

We say that Ω exhibits a $(k+1)$-*decrease* because $T^{k+1}(w) \sqsubset T^k(w)$. Notice that for a label chain to exhibit a $(k+1)$-decrease, the labels should belong to a lattice with at least one non-bottom element. Here, Ω' is a conservative approximation of Ω.

Consequently, whenever $(k+1)$-*Enf* initially associates flexible variable w with a label chain Ω that exhibits a $(k+1)$-decrease, enforcer k-*Enf* is forced by initialization condition ρ_k to use conservative approximation Ω' for Ω. So, as we prove in the Appendix, $(k+1)$-*Enf* is strictly more permissive than k-*Enf*.

Theorem 3. *k-Enf $<^{k,\mathcal{L}}_{\rho_k}$ $(k+1)$-Enf, for $k \geq 2$ and any lattice $\mathcal{L}$ with at least one non-bottom element.*

Thus, longer label chains offer increased permissiveness for the k-*Enf* family of enforcers. Moreover, we conclude by transitivity that k-*Enf* $<^{k,\mathcal{L}}_{\rho_k}$ ∞-*Enf*, for any $k \geq 2$.

There are cases where flexible variables initially store no information, and thus, they are initially associated with bottom-label chains (i.e., $\langle \bot, \ldots, \bot\rangle$). We say memory M is *conventionally initialized* when for set Var_F of flexible variables

$$\begin{aligned} \alpha_c(M) \triangleq \forall w \in \mathit{Var}_F\colon \forall i \geq 1\colon \\ T^i(w) \in \mathit{dom}(M) \Rightarrow M(T^i(w)) = \bot. \end{aligned}$$

We also define initialization condition

$$c(M, M') \triangleq \alpha_c(M) \wedge \rho_1(M, M')$$

which implies that two memories are conventionally initialized and agree on values in anchor variables and on the first labels of these anchor variables.

A result analogous to Theorem 3 does not hold when $<_{\rho_k}^{k,\mathcal{L}}$ is replaced with $<_{c}^{k,\mathcal{L}}$. With initialization condition c, label chains longer than two elements do not enhance the permissiveness of *k-Enf*. This is because, for initialization condition c, enforcer *k-Enf* produces label chains where the second element is always repeated[8] (e.g., $\langle \mathsf{H}, \mathsf{M}, \mathsf{M}, \ldots, \mathsf{M} \rangle$) due to the conservative update of label chains of flexible variable induced by rules (AsgnF) in Figure 3 and (Exit) in Figure 4. There, all elements of label chains of the involved flexible variables are updated with the same label (i.e., the sensitivity of the context). We prove the following theorem in the Appendix.

Theorem 4. *k-Enf* $\cong_{c}^{k,\mathcal{L}}$ *$(k+1)$-Enf for any lattice $\mathcal{L}$ and $k \geq 2$.*

Threat model specifics affect the permissiveness of longer label chains, too. Consider a *weakened threat model* that allows observations of updates to variables but not to tags. Enforcers here would be expected to satisfy 0-BNI. Enforcer *k-Enf* satisfies k-BNI. So, *k-Enf* satisfies 0-BNI, because 0-BNI is implied by k-BNI.

Under the weakened threat model, permissiveness of our enforcers is compared using relation $\leq_{\rho_k}^{0,\mathcal{L}}$, where superscript 0 indicates that only observations involving variables are considered for the comparison. Theorem 3 does not apply, because relation $<_{\rho_k}^{k,\mathcal{L}}$ considers observations up to the kth tag (due to superscript k) where $k \geq 2$. But we do have the following Theorem, which is proved in the Appendix.

Theorem 5. *k-Enf* $\cong_{\rho_k}^{0,\mathcal{L}}$ *$(k+1)$-Enf for any lattice $\mathcal{L}$ and $k \geq 2$.*

Because $c \Rightarrow \rho_k$ holds, property (17) and Theorem 5 gives *k-Enf* $\cong_{c}^{0,\mathcal{L}}$ $(k+1)$-*Enf* for any lattice $\mathcal{L}$ and $k \geq 2$. So, under the weakened threat model and for both initialization conditions (i.e, ρ_k and c), the permissiveness of *k-Enf* does not improve by using label chains of length greater than two.

Figure 7 summarizes the results presented in this section.

[8] See Lemma 11 in the Appendix.

		Initialization Condition	
		ρ_k	c
Threat Model	Strong	✓	x
	Weak	x	x

Figure 7: ✓ indicates enhanced permissiveness from label chains with more than two elements; x indicates no permissiveness gains for our family of k-*Enf* enforcers with $k \geq 2$.

8 Other Enforcers

We now relate permissiveness with label chain length for enforcers other than k-*Enf*. Here longer label chains might increase permissiveness. But the results depend on the threat model, lattice size, and certain semantic properties: k-*precise*, k-*varying*, and k-*dependent*.

8.1 In the Strong Threat Model

Longer label chains are useful for an enforcer E under the strong threat model provided there are executions of commands for which E produces label chains whose elements

(i) are not redundant—they are not a function of other elements in the same label chain, and

(ii) capture the real sensitivity of the elements they tag rather than conservatively approximating it.

Label chains that can be used as evidence for properties (i) and (ii) are characterized below as being k-*varying* and k-*precise*.

Label chains $\langle \ell_1, \ell_2, \ldots, \ell_k \rangle$ and $\langle \ell'_1, \ell'_2, \ldots, \ell'_k \rangle$ with labels from lattice $\mathcal{L}$, are defined to be k-varying for $k \geq 2$ iff

$$(\forall i\colon 1 \leq i < k\colon \ell_i = \ell'_i) \ \wedge \ \ell_k \neq \ell'_k.$$

Notice that k-varying label chains cannot exist when $\mathcal{L}$ has only one element.

We now formalize k-precise. Consider an enforcer E, lattice $\mathcal{L}$, command C, and conventionally initialized memory M such that $M \models \mathcal{H}_0(E, \mathcal{L}, C)$.

Assume trace $\tau = \mathit{trace}_E(C, M)$ *produces* label chain prefix $\Omega = \langle \ell_1, \ldots, \ell_n \rangle$ at some state $\tau[j]$ after an assignment to a flexible variable w:

$$\begin{aligned}
&\exists 1 < j \leq |\tau| \colon \exists w \in \mathit{Var}_F \colon \\
&\quad \tau[j-1] = \langle w := e; C_r, M_w \rangle \ \wedge\ \tau[j] = \langle C_r, M_r \rangle \ \wedge \\
&\quad \forall i \colon 1 \leq i \leq n \colon T^i(w) \in \mathit{dom}(M_r) \ \wedge\ M_r(T^i(w)) = \ell_i.
\end{aligned}$$

Label chain Ω is k-precise (for $1 \leq k \leq n$) at $\tau[j]$ when for each enforcer E':
if

- E' satisfies $(k-1)$-BNI($\mathcal{L}$), and
- $E \leq_c^{k-1,\mathcal{L}} E'$,

then

- trace $\tau' = \mathit{trace}_{E'}(C, M')$ with $M' \models \mathcal{H}_0(E', \mathcal{L}, C)$ and $c(M, M')$ produces label chain $\langle \ell_1, \ldots, \ell_k \rangle$ at $\tau'[j]$.

So, if Ω is k-precise, then any enforcer E' that satisfies $(k-1)$-BNI($\mathcal{L}$) and is at least as permissive as E (i.e., $E \leq_c^{k-1,\mathcal{L}} E'$) will produce (at the same execution point) the same first k elements that appear in Ω. Consequently, the first k elements of Ω capture the real sensitivity of the elements they tag.

For brevity, we say that E *produces some k-precise k-varying label chains* with elements in $\mathcal{L}$ iff there exist commands C, C' whose executions produce label chains Ω, Ω' such that:

- Ω is k-precise at the ith state of $\mathit{trace}_E(C, M)$, for some i and M with $M \models \mathcal{H}_0(E, \mathcal{L}, C)$,
- Ω' is k-precise at the jth state of $\mathit{trace}_E(C', M')$, for some j and M' with $M' \models \mathcal{H}_0(E, \mathcal{L}, C')$,
- Ω and Ω' are k-varying.

Longer label chains can offer increased permissiveness for an enforcer E, under the strong threat model, provided E produces some k-precise k-varying label chains. To see this, compare such an enforcer E with an enforcer E' that approximates the kth element of each label chain as a function of the previous elements instead of performing, for example, an analysis of the code.

We say that E' produces $(k-1)$-*dependent label chains* for $k-1 \geq 1$ iff E' is an enforcer and for some function $f_{E'}$:

$$\forall x\colon \forall i\colon k{-}1 < i < n_{E'}\colon T^i(x) = f_{E'}(T(x), \ldots, T^{k-1}(x))$$

For example, *k-Enf* produces k-dependent label chains, because *k-Enf* uses $f_{k\text{-}Enf}(T(x), \ldots, T^k(x)) \triangleq T^k(x)$ for computing $T^{k+1}(x)$. Notice, if an enforcer E' produces $(k-1)$-dependent label chains, then that mechanism cannot produce k-varying label chains.

Such an enforcer E' cannot both satisfy $(k-1)$-BNI and be at least as permissive as E, which produces some k-precise k-varying label chains: Assume for contradiction that E' satisfies $(k-1)$-BNI and is at least as permissive as E. Because the k-varying label chains produced by E are k-precise, E' should then produce the same k-varying label chains. But, we previously saw that if an enforcer E' produces k-varying label chains, then E' does not produce $(k-1)$-dependent label chains, which is a contradiction. A detailed proof of Theorem 6 is found in the Appendix.

Theorem 6. (i) *For a lattice $\mathcal{L}$, for an enforcer E that satisfies $(k-1)$-BNI($\mathcal{L}$), with $k \geq 2$, and produces some k-precise k-varying label chains with elements in $\mathcal{L}$, and for an enforcer E' that produces $(k-1)$-dependent label chains,*

$$\textit{if } E \preceq_c^{k-1,\mathcal{L}} E', \textit{ then } E' \textit{ does not satisfy } (k-1)\textit{-BNI}(\mathcal{L}).$$

(ii) *Enforcer E and lattice $\mathcal{L}$ exist.*

For an enforcer E' that uses label chains of length $k-1$ (i.e., produces $(k-1)$-dependent label chains), Theorem 6 implies that E' cannot be at least as permissive as an enforcer E that uses label chains of length k. So, in contrast to Theorem 4, which stipulates that *k-Enf* does not benefit from longer label chains under conventional initialization, enforcer E in Theorem 6 does benefit.

Theorem 6 (ii) asserts that such an enforcer E and lattice $\mathcal{L}$ exist. So, it is always possible to define, for each $k > 1$, an enforcer E that can produce k-precise k-varying label chains when executing some command C. Notice, *k-Enf* cannot produce k-precise k-varying label chains.

Witness E and $\mathcal{L}$ for Theorem 6 (ii) In the Appendix, we describe k-*Eopt*, which is an enforcer that satisfies $(k-1)$-BNI and produces some k-precise k-varying label chains during the execution of a certain command C. C involves sequences of assignments and **if** commands whose branches contain only one assignment. Such **if** commands will be called *simple*. We construct k-*Eopt* by optimizing k-*Enf* for deducing k-precise k-varying labels during the execution of such C. The optimization is based on the following observation: ignoring context, if $T^i(w) = \bot$ at the end of both branches of a simple **if** command, then, at the end of that **if** command, $T^{i+1}(w)$ does not need to be updated with the sensitivity $T(e)$ of the guard of that **if** command. This optimization enables k-*Eopt* to produce some k-precise k-varying label chains.

8.2 In the Weakened Threat Model

In the weakened threat model, label chains of length two can offer enhanced permissiveness compared to label chains of length one: the metalabel enables the decision to block assignment commands to be more permissive. (Previous theorems concerned label chains with at least two elements). To illustrate, it suffices to consider *anchor-tailed* commands, which are a sequence $C; C'$ of commands where C does not involve any assignment to anchor variables and C' is a sequence of assignments to anchor variables.

Let $G^E_{a:=e}$ denote the condition used by an enforcer E for blocking an assignment $a := e$ to anchor variable a when execution reaches state $\langle a := e; C', M'\rangle$ in a trace $\mathit{trace}_E(C, M)$. Boolean expression $G^E_{a:=e}$ is satisfied in a memory M' according to

$$\begin{aligned} M'(G^E_{a:=e}) \;\Leftrightarrow\; & \langle a := e; C', M'\rangle \to \langle C', M''\rangle \\ & \text{is a subtrace of } \mathit{trace}_E(C, M). \end{aligned}$$

For assignment $a := e$ in an anchor-tailed command, $G^E_{a:=e}$ may depend on label chains of variables in

- assignment $a := e$ itself (to capture explicit flows), and
- the context of that assignment (to capture implicit flows). By definition of anchor-tailed commands, such an assignment is not encapsulated in any conditional command, but it may follow other assignments to anchor variables. So, the context of $a := e$ only references variables mentioned in assignments to anchor variables that precede $a := e$.

Let $V_{a:=e}$ denote the above set of variables.

We say $G^E_{a:=e}$ is a *k-dependent condition* for $a := e$ in an anchor-tailed command iff $G^E_{a:=e}$ depends at most on the first k elements of the label chains of variables in $V_{a:=e}$

$$G^E_{a:=e} = f_E(\{T^i(x) \mid x \in V_{a:=e} \ \wedge \ 1 \leq i \leq k\}),$$

for some function f_E. For example, 2-*Enf* uses 2-dependent $G_{a:=e}$.

We now show how the second label in a label chain makes the decision to block assignment commands more permissive. Theorem 7, which is proved in the Appendix, states that if an enforcer E uses 1-dependent $G^E_{a:=e}$, then E cannot both satisfy 0-BNI and be at least as permissive as 2-*Enf*. Here is why. E does not compute the sensitivity of labels referenced by block condition $G^E_{a:=e}$ and thus E does not compute the sensitivity of the information conveyed by its decision to block a certain assignment $a := e$. In an effort to satisfy 0-BNI and prevent leaking sensitive information, E must decide always to block $a := e$, even though in some executions that assignment is safe and allowed by 2-*Enf*.

Theorem 7. *For an enforcer E and lattice $\mathcal{L}_3 \triangleq \langle\{\mathsf{L}, \mathsf{M}, \mathsf{H}\},\ \sqsubseteq,\ \sqcup\rangle$, if $G^E_{a:=e}$ is 1-dependent and 2-Enf $\leq_c^{0,\mathcal{L}_3} E$, then E does not satisfy 0-BNI($\mathcal{L}_3$).*

Thus, enforcer E that uses label chains of length one (i.e., $G^E_{a:=e}$ is 1-dependent) cannot be at least as permissive as 2-*Enf*, which uses label chains of length two (i.e., $G_{a:=e}$ is 2-dependent). So, for the weakened threat model, permissiveness can be improved when using two (instead of one) labels for each variable.

Since most dynamic enforcement mechanisms proposed in the past satisfy TINI, we might wonder whether Theorem 7 still holds when 0-BNI is replaced by TINI. Under the weakened threat model, there are enforcers (e.g., $E_{\mathsf{H},\mathsf{L}}$ in the next section) that use 1-dependent $G^E_{a:=e}$, are at least as permissive as 2-*Enf* and do satisfy TINI. So, Theorem 7 does not hold when 0-BNI is replaced by TINI.

Familiar Two-level Lattice

Some authors use a two-level lattice $\mathcal{L}_2 \triangleq \langle\{\mathsf{L}, \mathsf{H}\},\ \sqsubseteq,\ \sqcup\rangle$ with $\mathsf{L} \sqsubset \mathsf{H}$, believing that their results will extend to arbitrary lattices. In this section, we give a result for $\mathcal{L}_2$ that does not hold for more complex lattices. Thus, generalizing from $\mathcal{L}_2$ to arbitrary lattices is not always a sound proposition.

Consider $\mathcal{L}_2$ with the weakened threat model. Previous work [22] proposed a flow-sensitive enforcement mechanism that uses only one label per variable. We denote that enforcement mechanism by $E_{\mathsf{H},\mathsf{L}}$, which is derived from *k-Enf* by associating each variable with only one tag. Figure 8 shows the modified rules for $E_{\mathsf{H},\mathsf{L}}$. We prove below (Theorem 8) that $G_{a:=e}$ defined in Figure 8 is 1-dependent.

$E_{\mathsf{H},\mathsf{L}}$ ensures that the sensitivity of each tag $T(w)$ is always L, so there is no need to explicitly keep track of $T^2(w)$. The only way to encode information tagged with H in $T(w)$ is if $T(w)$ is updated with different labels in a conditional command that has a guard tagged with H. But, if the sensitivity of the guard is H, then due to function U in Figure 8, tag $T(w)$ will always be updated to H at the end of that conditional command, because $M(\lfloor cc \rfloor) = \mathsf{H}$. So, $T(w)$ will reveal no information about the value of that sensitive guard. Thus, the sensitivity of $T(w)$ is L.

Define function $trace_{E_{\mathsf{H},\mathsf{L}}}(C, M)$ to map command C and memory M with $M \models \mathcal{H}_0(E_{\mathsf{H},\mathsf{L}}, \mathcal{L}, C)$ to the entire trace that starts with state $\langle C, M \rangle$. We have $n_{E_{\mathsf{H},\mathsf{L}}} = 1$, $Aux_{E_{\mathsf{H},\mathsf{L}}} = \{cc, bc\}$, $Init_{E_{\mathsf{H},\mathsf{L}}}(cc) = \epsilon$, and $Init_{E_{\mathsf{H},\mathsf{L}}}(bc) = \bot$.

Theorem 7 does not hold when $\mathcal{L}_3$ is replaced with $\mathcal{L}_2$ if E is $E_{\mathsf{H},\mathsf{L}}$. Instead, Theorem 8 below holds; it states that $E_{\mathsf{H},\mathsf{L}}$ satisfies 0-BNI and is strictly more permissive than 2-*Enf* only when $\mathcal{L}_2$ is used.

Theorem 8. *Enforcer $E_{\mathsf{H},\mathsf{L}}$ uses 1-dependent $G_{a:=e}$, satisfies 0-BNI($\mathcal{L}_2$), and satisfies 2-Enf $<_c^{0,\mathcal{L}_2}$ $E_{\mathsf{H},\mathsf{L}}$.*

So Theorem 8, which is proved in the Appendix, contradicts expectations that longer label chains can offer increased permissiveness. Moreover, this theorem is an example where a result expressed in terms of $\mathcal{L}_2$ does not necessarily generalize for arbitrary lattices.

Notice, though, that $E_{\mathsf{H},\mathsf{L}}$ does not satisfy 0-BNI for arbitrary lattices. For example, consider (2), which employs $\mathcal{L}_3$. Based on rules in Figure 8 and rules (If),(Seq) in §5.3, $E_{\mathsf{H},\mathsf{L}}$ executes (2) as described in (i) and (ii) in §1. So, executing (2) under $E_{\mathsf{H},\mathsf{L}}$ leaks sensitive $m > 0$ to principals observing non-sensitive variable l, and thus, 0-BNI is not satisfied. $E_{\mathsf{H},\mathsf{L}}$ thus illustrates that an enforcer designed to enforce two-level lattices cannot necessarily enforce arbitrary lattices.

Figure 9 summarizes the results presented in this section. We do not have a proof but we conjecture that label chains with more than two elements do not improve permissiveness for lattices with more than two elements under the weakened threat model.

$$(\textsc{AsgnA})\ \frac{v = M(e) \qquad G_{a:=e} \qquad \ell = M(\lfloor cc \rfloor) \sqcup M(bc)}{\langle a := e, M\rangle \rightarrow \langle \mathbf{stop}, M[a \mapsto v, bc \mapsto \ell]\rangle}$$

$$(\textsc{AsgnAFail})\ \frac{v = M(e) \qquad \neg G_{a:=e} \qquad \ell = M(\lfloor cc \rfloor) \sqcup M(bc)}{\langle a := e, M\rangle \rightarrow \langle \mathbf{block}, M[bc \mapsto \ell]\rangle}$$

$$(\textsc{AsgnF})\ \frac{v_0 = M(e) \qquad v_1 = M(T(e)) \sqcup M(\lfloor cc \rfloor) \sqcup M(bc)}{\langle w := e, M\rangle \rightarrow \langle \mathbf{stop}, M[w \mapsto v_0, T(w) \mapsto v_1]\rangle}$$

$$U(M, W) \triangleq M[\,\forall w{\in}W\colon T(w) \mapsto T(w) \sqcup M(\lfloor cc \rfloor) \sqcup M(bc)]$$

$$G_{a:=e} \text{ is } M(T(e)) \sqcup M(\lfloor cc \rfloor) \sqcup M(bc) \sqsubseteq M(T(a))$$

Figure 8: Modified rules for $E_{\mathsf{H},\mathsf{L}}$

		Label Chain Length	
		Greater than 1	Greater than 2
Threat Model	Strong	✓	✓
	Weak ($\mathcal{L}_3$)	✓	?
	Weak ($\mathcal{L}_2$)	×	×

Figure 9: ✓ indicates where labels chains with length greater than the one indicated in the corresponding column can provide enhanced permissiveness; × indicates where longer label chains do not enhance permissiveness; ? indicates an open question.

9 Related Work

Dynamic Enforcement Mechanisms and Leaks The formalization of dynamic information flow enforcement mechanisms dates back to Bell and LaPadula [8]. The community realized early that dynamic enforcement mechanisms for information flow control might introduce leaks not present in the program itself. Denning [29], for instance, explains that blocking an execution and reporting the underlying violation might leak sensitive information. Denning also gives examples where flow-sensitive labels generated by dynamic enforcement mechanisms violate TINI. Our *k-Enf* enforcers do not report the reason an execution terminates for exactly this reason. Also they ensure that information is not leaked by observing flow-sensitive labels during normally terminated or blocked executions.

Label Chains of Length One Most dynamic enforcement mechanisms use label chains of length one. *Purely dynamic* enforcement mechanisms that analyze only code that is executed and employ *no-sensitive-upgrade* (NSU) or *permissive-upgrade* (PU) (e.g., [3], [4], [10], [15], [19], [33]) satisfy TINI but not BNI, because they leak sensitive information when blocking an execution. Other *hybrid* flow-sensitive enforcement mechanisms (e.g., [14], [26]), which employ program analysis before and during execution, do not satisfy BNI, either. There are enforcement mechanisms (e.g., [1], [7]) that satisfy BNI, but they either handle only $\mathcal{L}_2$ or lose permissiveness by tagging variables with the same labels at the end of conditional commands independent of which branch is actually taken. We are not aware of an enforcement mechanism that uses label chains of length one, enforces labels from an arbitrary lattice, satisfies BNI, and is at least as permissive as 2-*Enf*. We are also not aware of an enforcement mechanism that uses label chains of length one, enforces $\mathcal{L}_2$, satisfies BNI, and is at least as permissive as $E_{\mathsf{H},\mathsf{L}}$.

Label Chains of Length Two Certain dynamic enforcement mechanisms use label chains of length two. Buiras et al. [12] propose a purely dynamic enforcement mechanism that employs fixed metalabels to capture implicit flows caused by conditional commands. The purely dynamic enforcement mechanism in [12] causes insecure executions to diverge instead of blocking. By enforcing only TINI, no security guarantee is given for executions that are forced to diverge (because TINI considers only finite traces).

Bedford et al. [6] use label chains where the second element is flow sensitive. That hybrid enforcement mechanism enforces TSNI on programs written in a **while**-language that supports references. The enforcement mechanism uses 2-dependent label chains and, therefore, Theorem 6 implies that this enforcement mechanism is not more permissive than an enforcer that produces 3-precise 3-varying label chains (e.g., 3-*Eopt*).

Unbounded Label Chains Some enforcement mechanisms support label chains of unbounded length. Zheng et al. [38] employ dependent types to tag a label with another label, thus forming chains of labels. Their approach can express a label recursively tagging itself, which can be seen as infinitely repeating the last label of a chain. Examples presented in [38] employ label chains of up to two elements (e.g., $\langle \ell, \bot \rangle$ and $\langle \ell, \ell \rangle$), but the authors acknowledge [38, §3.3.2] that longer chains are sensible but do not show—as we do in this paper—that permissiveness can benefit from longer label chains. We explained (§7) why permissiveness can be lost when using label chains of fixed length (instead of using longer label chains).

The enforcement mechanism presented in Zheng et al. [38] is mostly static, so it does not exhibit the kinds of leaks our paper examines through flow sensitive labels and blocking executions. Specifically, label chains in [38] are given as input; they are not deduced by the enforcement mechanism. Conditions on these labels are inlined by the programmer. If the static analysis succeeds, then the program will satisfy TINI. So, a type-correct program can be safely executed until normal termination. Techniques presented in [38] involving label chains have been implemented in Jif [27, 28]. We believe that any framework that supports dependent types, such as [13] and [24], is likely capable of expressing unbounded label chains.

Actions Other than Blocking Dynamic enforcement mechanisms can take actions other than blocking when an unsafe command is about to be executed. Enforcement mechanisms presented in [14] and [23], which handle $\mathcal{L}_2$, modify or skip the execution of an unsafe command. Similar to [12], the enforcement mechanism presented in [25] (which enforces labels from $\mathcal{L}_2$) diverges when reaching an unsafe command. Some enforcement mechanisms (e.g., [9], [17],[18],[34]) take no action, because they only update labels on variables; they do not perform any checks.

Certain purely dynamic enforcement mechanisms (e.g., [20], [35]) recover from exceptions caused by unsafe commands. They enforce *error sensitive noninterference*, which we believe is stronger than BNI. One technique they employ is assigning the same labels to variables after conditional commands, independent of the branch that is taken. Here, some permissiveness might be lost against 2-*Enf*, which allows labels on variables to depend on taken branches.

Comparing Enforcement Mechanisms Russo et al. [31] study tradeoffs between static and dynamic security analysis. They prove impossibility of a purely dynamic information-flow monitor that satisfies TINI and accepts programs certified by the Hunt and Sands classical flow-sensitive static analysis [21]. They first define basic semantics that purely dynamic information-flow monitor may extend. Then, they introduce properties (i.e., *not look ahead*, *not look aside*) for the purely dynamic enforcement mechanisms they consider. Their impossibility theorem has the same style as our Theorem 7: an enforcement mechanism with the above properties cannot both satisfy TINI and be at least as permissive as [21]. Our Theorem 6 instead compares permissiveness of any two enforcement mechanisms that satisfy particular properties. And our permissiveness relation $\leq_{\rho}^{k,\mathcal{L}}$ is more general than the one presented in [31], because it is defined on any two enforcers and handles arbitrary lattices (not just $\mathcal{L}_2$) and initialization conditions.

Bielova et al. [11] present a taxonomy of five representative flow-sensitive information flow enforcement mechanisms (no-sensitive-upgrade, permissive-upgrade, hybrid monitor, secure multi-execution, and multiple facets), in terms of soundness, *precision*, and *transparency*, which stipulates that enforcement mechanisms do not alter the semantics of safe executions. *Termination Aware Noninterference* (TANI) is the soundness goal, and it is expressed in terms of knowledge semantics. If an enforcement mechanism diverges the execution of an unsafe command satisfies TANI, then this mechanism does not leak sensitive information by taking this action. The theoretical framework considered in [11] assumes labels are taken from $\mathcal{L}_2$. Also, it assumes that a terminating execution produces one output, at the end, tagged L; if an execution diverges, no output is produced. So, TANI guarantees that dynamic enforcement mechanisms do not introduce leaks when it diverges executions in the framework of [11]. Our section 8.2 explains that there is no danger these mechanisms could encode sensitive information in the flow-sensitive labels, because the framework in[11] is restricted to $\mathcal{L}_2$.

A Definitions

The following definitions are used in the proofs appearing in this appendix.

- We abbreviate $\tau|_{\ell}^{S}$ by $\tau|_{\ell}$, when S is the set of all variables and their tags.
- We extend the projection of a memory with respect to a label ℓ to include bc (blocking context) and cc (conditional context):

$$\begin{aligned}
&M|_{\ell} = \\
&\{\langle q, M(q)\rangle \mid q, T(q) \in dom(M) \ \wedge\ M(T(q)) \sqsubseteq \ell\} \cup \\
&\{\langle bc, M(bc)\rangle \mid M(bc) \sqsubseteq \ell\} \cup \\
&\{\langle cc, M(cc)\rangle \mid M(\lfloor cc \rfloor) \sqcup M(bc) \sqsubseteq \ell\}
\end{aligned} \tag{18}$$

- Define equality of sequences of observations when omitting the empty sets:

$$
\begin{aligned}
\epsilon &=_{obs} \epsilon \\
\emptyset &=_{obs} \epsilon \\
\epsilon &=_{obs} \emptyset \\
\Theta \rightharpoonup \theta &=_{obs} \Theta' \rightharpoonup \theta' \text{ iff} \\
&\begin{cases}
\Theta = \Theta' \ \wedge \ \theta =_{obs} \theta', \text{ or} \\
\Theta = \emptyset \ \wedge \ \Theta' \neq \emptyset \ \wedge \ \theta =_{obs} \Theta' \rightharpoonup \theta', \text{ or} \\
\Theta \neq \emptyset \ \wedge \ \Theta' = \emptyset \ \wedge \ \Theta \rightharpoonup \theta =_{obs} \theta'
\end{cases}
\end{aligned}
$$

- Define $kstut(M)$ to hold when the k^{th} label in all label chains in M is infinitely repeated:

$$
\begin{aligned}
kstut(M) \triangleq\ & \forall x{\in}dom(M) \colon \forall i > k \colon \\
& T^i(x){\in}dom(M) \Rightarrow M(T^i(x)) = M(T^k(x))
\end{aligned}
$$

  for $k \geq 1$

- We write $mon(M)$ to denote that all label chains in M are monotonically decreasing:

$$
\begin{aligned}
mon(M) \triangleq\ & \forall x{\in}dom(M) \colon \forall i \geq 1 \colon \\
& T^{i+1}(x){\in}dom(M) \Rightarrow M(T^{i+1}(x)) \sqsubseteq M(T^i(x))
\end{aligned}
$$

- We write $M =_k M'$ iff

  1. $\forall x \colon \forall 0 \leq i \leq k \colon M(T^i(x)) = M'(T^i(x))$,
  2. $M(cc) = M'(cc)$, and
  3. $M(bc) = M'(bc)$.

- To prove inductively that our enforcers satisfy BNI, we strengthen BNI to BNI+ and prove BNI+ instead.

  BNI+$(E, \mathcal{L}, C)$**.** $\forall \ell \in \mathcal{L} \colon \forall M, M' \colon$
  If

$$
\begin{aligned}
& M \models \mathcal{H}(E, \mathcal{L}, C) \ \wedge \ M' \models \mathcal{H}(E, \mathcal{L}, C), \\
& M|_\ell = M'|_\ell, M(cc) = M'(cc), mon(M), mon(M') \\
& \tau = trace_E(C, M), \\
& \tau' = trace_E(C, M'),
\end{aligned}
$$

where $\tau = \langle C, M\rangle \xrightarrow{*} \langle C_t, M_t\rangle$, $\tau' = \langle C, M'\rangle \xrightarrow{*} \langle C'_t, M'_t\rangle$, *and* C_t, C'_t *are terminations (i.e.,* **stop** *or* **block***),*
then:

*c*1 *If* C_t *and* C'_t *are both* **stop***, then* $\tau|_\ell =_{obs} \tau'|_\ell$, $M_t|_\ell = M'_t|_\ell$, *and* $M_t(cc) = M'_t(cc)$.

*c*2 *If* C_t *or* C'_t *is* **block***, then* $\tau|_\ell =_{obs} \tau'|_\ell$.

*c*3 *If* C_t *is* **stop***,* C'_t *is* **block***, and* $M'_t(bc) \not\sqsubseteq \ell$*, then* $M_t(bc) \not\sqsubseteq \ell$.

*c*4 *If* C_t *is* **stop***,* C'_t *is* **block***,* $\langle C'_{tp}, M'_{tp}\rangle \rightarrow \langle C'_t, M'_t\rangle$ *are the last two states of* τ'*, and* $M'_{tp}(\lfloor cc\rfloor) \sqsubseteq \ell$*, then there exists* $\langle C'', M''\rangle \in \tau$*, with* $C'' = C'_{tp}$ *and* $M''|_\ell = M'_{tp}|_\ell$.

- For enforcers k-*Enf* with $2 \leq k \leq \infty$, $E_{\mathsf{H,L}}$, and k-*Eopt* with $k \geq 2$, we extend the domain of function $trace_E(C, M)$ to also include any memory M such that $M \models \mathcal{H}(E, \mathcal{L}, C)$ holds.
  Also, if $M \models \mathcal{H}(E, \mathcal{L}, C)$ holds, then for any M' in $trace_E(C, M)$ and any subcommand C' of C, we have $M' \models \mathcal{H}(E, \mathcal{L}, C')$ (based on the corresponding operational semantics).

B Soundness of ∞-*Enf* and k-*Enf* for $k \geq 2$

Theorem 1. *∞-Enf is an enforcer on R for BNI.*

Proof. It is easy to prove that ∞-*Enf* is an enforcer on R and satisfies restrictions (E1), (E2), and (E3) by induction on the rules of ∞-*Enf*. We omit the details.

To prove that ∞-*Enf* satisfies BNI, we will prove that k-BNI(∞-*Enf*, $\mathcal{L}, C$) holds for a lattice $\mathcal{L}$, a command C, and $k \geq 0$. From Lemma 1, we have that BNI+(∞-*Enf*,$\mathcal{L}$,C) holds. By definition, $M \models \mathcal{H}_0(\infty\text{-}Enf, \mathcal{L}, C)$ and $M' \models \mathcal{H}_0(\infty\text{-}Enf, \mathcal{L}, C)$ imply $M(cc) = M'(cc), mon(M), mon(M')$. Also, $\tau|_\ell =_{obs} \tau'|_\ell$ implies $\tau|_\ell^k =_{obs} \tau'|_\ell^k$. Thus, BNI+($\infty$-*Enf*,$\mathcal{L}$,$C$) implies k-BNI(∞-*Enf*, $\mathcal{L}, C$). Because k, C, and $\mathcal{L}$ were arbitrary, we then get that ∞-*Enf* enforces BNI. □

Lemma 1. *For a command C and lattice $\mathcal{L}$, BNI+(∞-Enf,$\mathcal{L}$,C) holds.*

Proof. Let $\ell \in \mathcal{L}$. We use structural induction on C.

1. C is **skip**:
From rule Skip, we get $C_t = C'_t = \textbf{stop}$. So, $c2$, $c3$, and $c4$ are trivially true.

We prove $c1$. We have, $\tau|_\ell =_{obs} \epsilon$ and $\tau'|_\ell =_{obs} \epsilon$. Because $M_t = M$ and $M'_t = M'$, we get $M_t|_\ell = M'_t|_\ell$ and $M_t(cc) = M'_t(cc)$. So, $c1$ holds.

2. C is $a := e$:
From $M|_\ell = M'|_\ell$ and $M(T^2(a)) = M'(T^2(a)) = \bot$, we get

$$M(T(a)) = M'(T(a)). \tag{19}$$

2.1. $M(T(a)) \sqsubseteq \ell$
We first prove that the command is executed normally in both memories or blocked in both memories. W.l.o.g, assume that the command is executed normally in M. We prove that the command is executed normally in M', too. Because the command is executed normally in M, rule AsgnA in Figure 3 has been triggered, meaning that $M(T(e)) \sqcup M(\lfloor cc \rfloor) \sqcup M(bc) \sqsubseteq M(T(a))$ holds. From hypothesis (*2.1.*), we then get $M(T(e)) \sqsubseteq \ell$, $M(\lfloor cc \rfloor) \sqsubseteq \ell$, and $M(bc) \sqsubseteq \ell$. Because $M|_\ell = M'|_\ell$, we then get $M(cc) = M'(cc)$ and $M(bc) = M'(bc)$. From $mon(M)$ and $M(T(e)) \sqsubseteq \ell$, we get $M(T^2(e)) \sqsubseteq \ell$ and $M(T^3(e)) \sqsubseteq \ell$. From Lemma 5 and $M|_\ell = M'|_\ell$, we then get $M(T^2(e)) = M'(T^2(e))$, $M(T(e)) = M'(T(e))$, $M(e) = M'(e)$. From (19), we then get that $M'(T(e)) \sqcup M'(\lfloor cc \rfloor) \sqcup M'(bc) \sqsubseteq M'(T(a))$ holds. So, rule AsgnA is triggered, meaning that the command is executed normally in M'. Taking the contrapositive of the statement we just proved (i.e., if the command is executed normally in M, then it will be executed normally in M'), we get that if the command is blocked in M', then it will be blocked in M. Because M and M' are arbitrary, we consequently have that the command is either (i) executed normally in both memories or (ii) blocked in both memories. So, $c3$ and $c4$ are trivially true. To prove $c1$ and $c2$, we examine cases (i) and (ii).

2.1.1. The command is executed normally in both memories.
$c2$ is trivially true.
We prove $c1$. We have:
$\tau = \langle a := e, \mathcal{M} \rangle \rightarrow \langle \textbf{stop}, M[a \mapsto M(e), bc \mapsto \ell_g] \rangle$
$\tau' = \langle a := e, \mathcal{M}' \rangle \rightarrow \langle \textbf{stop}, M'[a \mapsto M'(e), bc \mapsto \ell'_g] \rangle$,

where

$$\ell_g = M(T^2(e)) \sqcup M(\lfloor cc \rfloor) \sqcup M(bc) \text{ and}$$
$$\ell'_g = M'(T^2(e)) \sqcup M'(\lfloor cc \rfloor) \sqcup M'(bc).$$

We have $\tau|_\ell = \tau'|_\ell = \langle a, M(e)\rangle$, because $M(e) = M'(e)$. Also, $\ell_g = \ell'_g$. So, $M_t(bc) = M'_t(bc)$. Because $M_t(cc) = M(cc) = M'(cc) = M'_t(cc)$, then $M_t(cc) = M'_t(cc)$ holds. Because $M_t(a) = M'_t(a)$, $M_t(bc) = M'_t(bc)$, and $M_t(cc) = M'_t(cc)$, then we get $M_t|_\ell = M'_t|_\ell$. So $c1$ holds.

2.1.2. The command is blocked in both memories.
$\tau = \langle a := e, \mathcal{M}\rangle \to \langle \textbf{block}, M[bc \mapsto \ell_g]\rangle$
$\tau' = \langle a := e, \mathcal{M}'\rangle \to \langle \textbf{block}, M'[bc \mapsto \ell'_g]\rangle$.
So, $\tau|_\ell = \tau'|_\ell = \epsilon$, and thus $c2$ holds. Also, $c1$ is trivially true.

2.2. $M(T(a)) \not\sqsubseteq \ell$

We then have $\tau|_\ell =_{obs} \epsilon$ and $\tau'|_\ell =_{obs} \epsilon$, and thus $c2$ holds.

To prove $c1$, assume $C_t = C'_t = \textbf{stop}$. We show $M_t(cc) = M'_t(cc)$ and $M_t|_\ell = M'_t|_\ell$.

Because $M(cc) = M'(cc)$ and $M'_t(cc) = M'(cc)$, we have $M_t(cc) = M'_t(cc)$.

We now prove $M_t|_\ell = M'_t|_\ell$ as per (18). Because $M(T(a)), M'(T(a)) \not\sqsubseteq \ell$, then M_t and M'_t do not need to agree on a. If $M_t(bc) \not\sqsubseteq \ell$, then $M_t|_\ell = M'_t|_\ell$ trivially holds. Assume instead $M_t(bc) \sqsubseteq \ell$. So, rule AsgnA in Figure 3 then gives $M(T^2(e)) \sqcup M(\lfloor cc \rfloor) \sqcup M(bc) \sqsubseteq \ell$. Thus, $M(T^2(e)) \sqsubseteq \ell$, $M(\lfloor cc \rfloor) \sqsubseteq \ell$, and $M(bc) \sqsubseteq \ell$. From $mon(M)$ and $M(T^2(e)) \sqsubseteq \ell$, we get $M(T^3(e)) \sqsubseteq \ell$. From $M|_\ell = M'|_\ell$, $M(T^3(e)) \sqsubseteq \ell$, and Lemma 5, we get $M(T^2(e)) = M'(T^2(e))$. From $M|_\ell = M'|_\ell$, $M(\lfloor cc \rfloor) \sqsubseteq \ell$, and $M(bc) \sqsubseteq \ell$, we get: $M(\lfloor cc \rfloor) = M'(\lfloor cc \rfloor)$ and $M(bc) = M'(bc)$. Because AsgnA in Figure 3 gives $M_t(bc) = M(T^2(e)) \sqcup M(\lfloor cc \rfloor) \sqcup M(bc)$ and $M'_t(bc) = M'(T^2(e)) \sqcup M'(\lfloor cc \rfloor) \sqcup M'(bc)$, we then have $M_t(bc) = M'_t(bc)$. From $M_t(cc) = M'_t(cc)$, we then get $M_t|_\ell = M'_t|_\ell$. So, $c1$ holds.

To prove $c3$, assume C_t is **stop**, C'_t is **block**, and $M'_t(bc) \not\sqsubseteq \ell$. We must show $M_t(bc) \not\sqsubseteq \ell$. We show the contrapositive. Assume $M_t(bc) \sqsubseteq \ell$, then following the same arguments as above, we get $M_t(bc) = M'_t(bc)$, and thus, $M'_t(bc) \sqsubseteq \ell$, as wanted. So, $c3$ holds.

To prove $c4$, assume C_t is **stop**, C'_t is **block**, $\langle C'_{tp}, M'_{tp}\rangle \to \langle C'_t, M'_t\rangle$ are the last two states of τ', and $M'_{tp}(\lfloor cc \rfloor) \sqsubseteq \ell$. So, $M'_{tp} = M'$ and

$C'_{tp} = a := e$. We have that $\langle C'', M'' \rangle$ in $c4$ is $\langle a := e, M \rangle$, which satisfies $C'' = C'_{tp}$ and $M''|_\ell = M'_{tp}|_\ell$. Thus $c4$ holds.

3. C is $w := e$
$\tau = \langle w := e, M \rangle \rightarrow \langle \textbf{stop}, M_t \rangle$
$\tau' = \langle w := e, M' \rangle \rightarrow \langle \textbf{stop}, M'_t \rangle$
$c2$, $c3$, and $c4$ are trivially true.

   We prove $c1$. $M_t(cc) = M'_t(cc)$ holds because we have $M(cc) = M'(cc)$, $M_t(cc) = M(cc)$, and $M'_t(cc) = M'(cc)$.

   3.1. $\exists r \geq 1$: $M_t(T^r(w)) \sqsubseteq \ell$
   From $mon(M)$, we then get $\forall i \geq r$: $M_t(T^i(w)) \sqsubseteq \ell$. Then we have $\forall i \geq r : M(T^i(e)) \sqsubseteq \ell$, $M(\lfloor cc \rfloor) \sqsubseteq \ell$, and $M(bc) \sqsubseteq \ell$. From, $M|_\ell = M'|_\ell$ and Lemma 5, we then get $\forall i \geq r-1 : M(T^i(e)) = M'(T^i(e))$, $M(cc) = M'(cc)$, and $M(bc) = M'(bc)$. So, $\forall i \geq r-1 : M_t(T^i(w)) = M'_t(T^i(w))$, and thus, $\forall i \geq r$: $M'_t(T^i(w)) \sqsubseteq \ell$.

      - If $r = 1$, we then get $M_t|_\ell = M'_t|_\ell$ and $\tau|_\ell = \tau'|_\ell$. Thus $c1$ holds.
      - Assume $r > 1$ holds. Then we have $\forall i$: $1 < i < r$: $M_t(T^i(w)) \not\sqsubseteq \ell$. Because $\forall i \geq r-1 : M_t(T^i(w)) = M'_t(T^i(w))$, we then get $M'_t(T^{r-1}(w)) \not\sqsubseteq \ell$. From $mon(M')$ we then get $\forall i < r :$ $M'_t(T^i(w)) \not\sqsubseteq \ell$. Thus, $M_t|_\ell = M'_t|_\ell$ and $\tau|_\ell = \tau'|_\ell$. Thus $c1$ holds.

   3.2. $\forall i \geq 1$: $M_t(T^i(w)) \not\sqsubseteq \ell$
   By symmetry of preceding case, $\forall i \geq 1$: $M'_t(T^i(w)) \not\sqsubseteq \ell$. So,$\tau|_\ell =_{obs} \epsilon$ and $\tau'|_\ell =_{obs} \epsilon$. Because $\forall i \geq 1$: $M_t(T^i(w)) \not\sqsubseteq \ell$ and $\forall i \geq 1$: $M'_t(T^i(w)) \not\sqsubseteq \ell$ holds, we get $M_t|_\ell = M'_t|_\ell$. Thus $c1$ holds.

4. $C_1; C_2$

   4.1. C_1 terminates normally in τ and τ'.
   Let
   $\tau = \langle C_1; C_2, M \rangle \xrightarrow{*} \langle C_2, M_2 \rangle \xrightarrow{*} \langle C_{2t}, M_t \rangle$ and
   $\tau' = \langle C_1; C_2, M' \rangle \xrightarrow{*} \langle C_2, M'_2 \rangle \xrightarrow{*} \langle C'_{2t}, M'_t \rangle$.
   Consider:
   $\tau_1 = \langle C_1, M \rangle \xrightarrow{*} \langle \textbf{stop}, M_2 \rangle$,
   $\tau_2 = \langle C_2, M_2 \rangle \xrightarrow{*} \langle C_{2t}, M_t \rangle$,
   $\tau'_1 = \langle C_1, M' \rangle \xrightarrow{*} \langle \textbf{stop}, M'_2 \rangle$,
   $\tau'_2 = \langle C_2, M'_2 \rangle \xrightarrow{*} \langle C'_{2t}, M'_t \rangle$.

From $c1$ of IH on C_1, we get $\tau_1|_\ell =_{obs} \tau_1'|_\ell$, $M_2|_\ell = M_2'|_\ell$, and $M_2(cc) = M_2'(cc)$. From $mon(M)$, $mon(M')$ and Lemma 2, we get $mon(M_2)$, $mon(M_2')$. So, we can apply IH on C_2.

To prove $c1$, assume $C_{2t} = C_{2t}' = \mathbf{stop}$. From IH on C_2, we get $\tau_2|_\ell =_{obs} \tau_2'|_\ell$, $M_t|_\ell = M_t'|_\ell$, and $M_t(cc) = M_t'(cc)$. From $\tau_1|_\ell =_{obs} \tau_1'|_\ell$ and $\tau_2|_\ell =_{obs} \tau_2'|_\ell$, we get $\tau|_\ell =_{obs} \tau'|_\ell$. Thus $c1$ holds.

To prove $c2$, assume $C_{2t} = \mathbf{block}$ or $C_{2t}' = \mathbf{block}$. From IH on C_2, we get $\tau_2|_\ell =_{obs} \tau_2'|_\ell$. From $\tau_1|_\ell =_{obs} \tau_1'|_\ell$ and $\tau_2|_\ell =_{obs} \tau_2'|_\ell$, we get $\tau|_\ell =_{obs} \tau'|_\ell$. Thus $c2$ holds.

To prove $c3$, assume $C_{2t} = \mathbf{stop}$, $C_{2t}' = \mathbf{block}$, and $M_t'(bc) \not\sqsubseteq \ell$. From IH on C_2, we get $M_t(bc) \not\sqsubseteq \ell$. Thus $c3$ holds.

To prove $c4$, assume $C_{2t} = \mathbf{stop}$, $C_{2t}' = \mathbf{block}$, $\langle C_{tp}', M_{tp}'\rangle \to \langle C_{2t}', M_t'\rangle$ are the last two states of τ', and $M_{tp}'(\lfloor cc \rfloor) \sqsubseteq \ell$. Then $\langle C_{tp}', M_{tp}'\rangle \to \langle C_{2t}', M_t'\rangle$ are the last two states of τ_2'. From IH on C_2, we get that there exists $\langle C'', M''\rangle \in \tau_2$, with $C'' = C_{tp}'$ and $M''|_\ell = M_{tp}'|_\ell$. Thus, there exists $\langle C'', M''\rangle \in \tau$, with $C'' = C_{tp}'$ and $M''|_\ell = M_{tp}'|_\ell$. Thus $c4$ holds.

4.2. C_1 is blocked in both τ, τ'

Similar to the above case, we apply IH on C_1.

4.3. C_1 is blocked in τ, terminates normally in τ' (C_2 may term/block in τ').

$c1$ is trivially true.

Let:

$\tau = \langle C_1; C_2, M\rangle \xrightarrow{*} \langle C_{1t}; C_2, M_t\rangle$ and

$\tau' = \langle C_1; C_2, M'\rangle \xrightarrow{*} \langle C_2, M_2'\rangle \xrightarrow{*} \langle C_{2t}', M_t'\rangle$.

Consider:

$\tau_1 = \langle C_1, M\rangle \xrightarrow{*} \langle C_{1t}, M_t\rangle$,

$\tau_1' = \langle C_1, M'\rangle \xrightarrow{*} \langle \mathbf{stop}, M_2'\rangle$,

$\tau_2' = \langle C_2, M_2'\rangle \xrightarrow{*} \langle C_{2t}', M_t'\rangle$.

So, we have

$$\tau|_\ell = \tau_1|_\ell \text{ and } \tau'|_\ell = \tau_1'|_\ell \rightharpoonup \tau_2'|_\ell. \tag{20}$$

We prove $c3$. We have $C_t = C_{1t} = \mathbf{block}$. Assume $C_t' = C_{2t}' = \mathbf{stop}$ and $M_t(bc) \not\sqsubseteq \ell$. We prove $M_t'(bc) \not\sqsubseteq \ell$. For IH on C_1, we get $M_2'(bc) \not\sqsubseteq \ell$. From Lemma 7, we then get $M_t'(bc) \not\sqsubseteq \ell$. Thus $c3$ holds.

We prove $c4$. We have $C_t = \mathbf{block}$. Assume $C_{2t}' = \mathbf{stop}$, that $\langle C_{tp}; C_2, M_{tp}\rangle \to \langle C_{1t}; C_2, M_t\rangle$ are the last two states of τ, and that

$M_{tp}(\lfloor cc \rfloor) \sqsubseteq \ell$ holds. Then $\langle C_{tp}, M_{tp}\rangle \to \langle C_{1t}, M_t\rangle$ are the last two states of τ_1. From IH on C_1, we get that there exists $\langle C'', M''\rangle \in \tau_1'$, with $C'' = C_{tp}$ and $M''|_\ell = M_{tp}|_\ell$. Thus, there exists $\langle C''; C_2, M''\rangle \in \tau'$, with $C'' = C_{tp}$ and $M''|_\ell = M_{tp}|_\ell$. Thus $c4$ holds.

We prove $c2$. From IH on C_1, we get we get $\tau_1|_\ell =_{obs} \tau_1'|_\ell$. Given also (20), to prove $\tau|_\ell =_{obs} \tau'|_\ell$, it suffices that $\tau_2'|_\ell = \epsilon$. So, we prove $\tau_2'|_\ell = \epsilon$. Assume the last transition of τ_1 is $\langle C_{1tp}, M_{tp}\rangle \to \langle C_{1t}, M_t\rangle$. τ_1 is blocked due to AsgnAFail, so C_{1tp} is $a := e; C'$.

4.3.1. $M_{tp}(bc) \not\sqsubseteq \ell$

From Lemma 7 and $\langle C_{1tp}, M_{tp}\rangle \to \langle C_{1t}, M_t\rangle$, we have $M_{tp}(bc) \sqsubseteq M_t(bc)$. Hypothesis (*4.3.1.*) then gives $M_t(bc) \not\sqsubseteq \ell$. From IH[$c3$] on C_1, we get $M_2'(bc) \not\sqsubseteq \ell$. From Lemma 4, $\tau_2'|_\ell = \epsilon$. Thus $c2$ holds.

4.3.2. $M_{tp}(bc) \sqsubseteq \ell$ and $M_{tp}(cc) \sqsubseteq \ell$

From IH[$c4$] on C_1, there exists $\langle C_{1tp}, M_1'\rangle \in \tau_1'$ such that $M_{tp}|_\ell = M_1'|_\ell$. So, $M_{tp}(bc) = M_1'(bc)$ and $M_{tp}(cc) = M_1'(cc)$. Because τ_1 is blocked, we have $M_{tp}(T(e)) \sqcup M_{tp}(\lfloor cc \rfloor) \sqcup M_{tp}(bc) \not\sqsubseteq M_{tp}(T(a))$. Since the inequality is satisfied in τ_1', it means that the value of $T(e)$ is different in M_1' and M_{tp}. So, $M_1'(T^2(e)) \not\sqsubseteq \ell$. Consider $\langle C_1, M'\rangle \xrightarrow{*} \langle C_{1tp}, M_1'\rangle \to \langle C_n, M_n\rangle$ a prefix of τ_1'. From AsgnA, we then have $M_n(bc) \not\sqsubseteq \ell$. From Lemma 7, we then get $M_2'(bc) \not\sqsubseteq \ell$. From Lemma 4, we then have $\tau_2'|_\ell = \epsilon$. Thus $c2$ holds.

4.3.3. $M_{tp}(bc) \sqsubseteq \ell$ and $M_{tp}(\lfloor cc \rfloor) \not\sqsubseteq \ell$

From AsgnAFail, $M_{tp}(\lfloor cc \rfloor) \sqsubseteq M_t(bc)$. So, $M_t(bc) \not\sqsubseteq \ell$. We work similarly to case *4.3.1.*.

5. **if** e **then** C_1 **else** C_2 **end**

5.1. $M(\lfloor cc \rfloor) \sqsubseteq \ell$ and $M(T(e)) \sqsubseteq \ell$

From $mon(M)$ and $M(T(e)) \sqsubseteq \ell$, we have $M(T^2(e)) \sqsubseteq \ell$. Because $M|_\ell = M'|_\ell$ and Lemma 5, we then have $M(T(e)) = M'(T(e))$ and $M(e) = M'(e)$. So, τ and τ' get the same branch, say C_1.

$\tau = \langle \textbf{if } e \textbf{ then } C_1 \textbf{ else } C_2 \textbf{ end}, M\rangle \to \langle C_1; \textbf{exit}, M_1\rangle \xrightarrow{*} \langle C_t, M_t\rangle$

$\tau' = \langle \textbf{if } e \textbf{ then } C_1 \textbf{ else } C_2 \textbf{ end}, M'\rangle \to \langle C_1; \textbf{exit}, M_1'\rangle \xrightarrow{*} \langle C_t', M_t'\rangle$.

From $M_1(cc) = M(cc).push(\langle M(T(e)), W, A\rangle)$, $M(cc) = M'(cc)$, $M(T(e)) = M'(T(e))$, and $M_1'(cc) = M'(cc).push(\langle M'(T(e)), W, A\rangle)$, we get $M_1(cc) = M_1'(cc)$.

We prove $M_1|_\ell = M_1'|_\ell$ as per (18). Because $M|_\ell = M'|_\ell$, it is trivially true when $M_1(bc) \not\sqsubseteq \ell$. Assume $M_1(bc) \sqsubseteq \ell$. Because $M_1(cc) = M_1'(cc)$ holds, it suffices to also prove $M_1(bc) = M_1'(bc)$. Because $M_1(bc) = M(bc)$, we then get $M(bc) \sqsubseteq \ell$. From $M|_\ell = M'|_\ell$, we then get $M(bc) = M'(bc)$. Because $M_1'(bc) = M'(bc)$, we then get $M_1(bc) = M_1'(bc)$. So, $M_1|_\ell = M_1'|_\ell$.

From $mon(M)$, $mon(M')$ and Lemma 2, we get $mon(M_1)$, $mon(M_1')$. We apply IH on $C_1;$ **exit** and get $c1$, $c2$, $c3$, and $c4$.

5.2. $M(\lfloor cc \rfloor) \not\sqsubseteq \ell$ or $M(T(e)) \not\sqsubseteq \ell$

We first prove that $M'(\lfloor cc \rfloor) \not\sqsubseteq \ell$ or $M'(T(e)) \not\sqsubseteq \ell$ holds. If $M(T(e)) \not\sqsubseteq \ell$, then from $M|_\ell = M'|_\ell$ and Lemma 6, we get $M'(T(e)) \not\sqsubseteq \ell$. Now, if $M(\lfloor cc \rfloor) \not\sqsubseteq \ell$, then $M(cc) = M'(cc)$ gives $M'(\lfloor cc \rfloor) \not\sqsubseteq \ell$. Thus, we have $M'(\lfloor cc \rfloor) \not\sqsubseteq \ell$ or $M'(T(e)) \not\sqsubseteq \ell$.

So, Lemma 3 gives:

(i) $\tau|_\ell =_{obs} \epsilon$ and $\tau'|_\ell =_{obs} \epsilon$.

(ii) If $C_t =$ **stop** (or $C_t' =$ **stop**), and $w \in \mathit{targetFlex}(C)$, then $\forall i : M_t(T^i(w)) \not\sqsubseteq \ell$ (or $\forall i : M_t'(T^i(w)) \not\sqsubseteq \ell$).

(iii) If $C_t =$ **stop** (or $C_t' =$ **stop**), and $\mathit{targetAnchor}(C) \neq \emptyset$ then $M_t(bc) \not\sqsubseteq \ell$ or $(M_t'(bc) \not\sqsubseteq \ell)$.

So, $c2$ holds.

We prove $c1$. Assume $C_t = C_t' =$ **stop**. Because $\tau|_\ell =_{obs} \epsilon$ and $\tau'|_\ell =_{obs} \epsilon$, it suffices to prove $M_t|_\ell = M_t'|_\ell$ and $M_t(cc) = M_t'(cc)$. From Lemma 8, we get $M(cc) = M_t(cc)$ and $M'(cc) = M_t'(cc)$. From $M(cc) = M'(cc)$, we then get $M_t(cc) = M_t'(cc)$. We prove $M_t|_\ell = M_t'|_\ell$. If $M_t(T^{i+1}(x)) \sqsubseteq \ell$, then (ii) gives $x \notin \mathit{targetFlex}(C)$. So, $M_t(T^i(x)) = M(T^i(x))$, $M'(T^i(x)) = M_t'(T^i(x))$, $M_t(T^{i+1}(x)) = M(T^{i+1}(x))$, $M'(T^{i+1}(x)) = M_t'(T^{i+1}(x))$. Thus, $M(T^{i+1}(x)) \sqsubseteq \ell$. From $M|_\ell = M'|_\ell$, we then have $M'(T^{i+1}(x)) \sqsubseteq \ell$ and $M(T^i(x)) = M'(T^i(x))$. By transitivity, $M_t(T^i(x)) = M_t'(T^i(x))$ and $M_t'(T^{i+1}(x)) \sqsubseteq \ell$. Assume $M_t(bc) \sqsubseteq \ell$. So, (iii) gives $\mathit{targetAnchor}(C) = \emptyset$. So, $M(bc) = M_t(bc)$ and $M'(bc) = M_t'(bc)$. From $M_t(bc) \sqsubseteq \ell$, we then get $M(bc) \sqsubseteq \ell$. From $M|_\ell = M'|_\ell$, we then get $M(bc) = M'(bc)$. Thus, $M_t(bc) = M_t'(bc)$. So, $M_t|_\ell = M_t'|_\ell$. Thus $c1$ holds.

We prove $c3$. If $C_t' =$ **block**, then $\mathit{targetAnchor}(C) \neq \emptyset$. So, (iii) gives $M_t(bc) \not\sqsubseteq \ell$. Thus $c3$ holds.

We prove $c4$. We have that $M'(\lfloor cc \rfloor) \not\sqsubseteq \ell$ or $M'(T(e)) \not\sqsubseteq \ell$. So, if $C'_t = \mathbf{block}$, then $M'_{tp}(\lfloor cc \rfloor) \not\sqsubseteq \ell$. Thus $c4$ holds.

6. **while** e **do** C_1 **end**
Induction on the maximum number of iterations in τ and τ'.

Base case: Both τ and τ' take (Wl2).
$\tau = \langle \mathbf{while}\ e\ \mathbf{do}\ C_1\ \mathbf{end}, \mathcal{M}\rangle \to \langle \mathbf{exit}, M_e\rangle \to \langle \mathbf{stop}, M_t\rangle$
$\tau' = \langle \mathbf{while}\ e\ \mathbf{do}\ C_1\ \mathbf{end}, \mathcal{M}'\rangle \to \langle \mathbf{exit}, M'_e\rangle \to \langle \mathbf{stop}, M'_t\rangle$.
So, $c2$, $c3$, $c4$ are trivially true.

We prove $c1$. We have $\tau|_\ell =_{obs} \epsilon$ and $\tau'|_\ell =_{obs} \epsilon$.

6.1. $M(T(e)) \sqsubseteq \ell$:

We prove $M_e(cc) = M'_e(cc)$. From $M(T(e)) \sqsubseteq \ell$ and $mon(M)$, we then have $M(T^2(e)) \sqsubseteq \ell$. From $M|_\ell = M'|_\ell$ and Lemma 5, we then get $M(T(e)) = M'(T(e))$. Both τ and τ' have the same W and A. So, from $M(cc) = M'(cc)$ and $M(T(e)) = M'(T(e))$, we then get $M_e(cc) = M'_e(cc)$.

We now prove $M_e|_\ell = M'_e|_\ell$. If $M_e(bc) \not\sqsubseteq \ell$, then it is trivial, given $M|_\ell = M'|_\ell$. Assume $M_e(bc) \sqsubseteq \ell$. Given $M_e(cc) = M'_e(cc)$, it suffices to prove $M_e(bc) = M'_e(bc)$. We have $M(bc) = M_e(bc)$ and $M'(bc) = M'_e(bc)$. Because $M_e(bc) \sqsubseteq \ell$, we then get $M(bc) \sqsubseteq \ell$. From $M|_\ell = M'|_\ell$, we then get $M(bc) = M'(bc)$. Because $M(bc) = M_e(bc)$ and $M'(bc) = M'_e(bc)$, we then get by transitivity $M_e(bc) = M'_e(bc)$.

So, $M_e(cc) = M'_e(cc)$ and $M_e|_\ell = M'_e|_\ell$. From Lemma 2, $mon(M)$, and $mon(M')$, we get $mon(M_e)$ and $mon(M_e)$. We use the proof for **exit** to get $M_t|_\ell = M'_t|_\ell$ and $M_t(cc) = M'_t(cc)$. Thus $c1$ holds.

6.2. $M(T(e)) \not\sqsubseteq \ell$:

In case (*6.1.*), we showed that $M(T(e)) \sqsubseteq \ell$ implies $M(T(e)) = M'(T(e))$, which gives $M'(T(e)) \sqsubseteq \ell$. The contrapositive of this statement is that $M'(T(e)) \not\sqsubseteq \ell$ gives $M(T(e)) \not\sqsubseteq \ell$. Because M, M' are arbitrary and because $M(T(e)) \not\sqsubseteq \ell$ (hypothesis of this case), we then get $M'(T(e)) \not\sqsubseteq \ell$.

We prove $M_t(cc) = M'_t(cc)$. From Lemma 8, we get $M(cc) = M_t(cc)$ and $M'(cc) = M'_t(cc)$. From $M(cc) = M'(cc)$, we then get $M_t(cc) = M'_t(cc)$.

We prove $M_t|_\ell = M'_t|_\ell$. Using Lemma 3, if $M_t(T^{i+1}(x)) \sqsubseteq \ell$, then $x \notin$ *targetFlex*(C). So, $M_t(T^i(x)) = M(T^i(x))$, $M'(T^i(x)) = M'_t(T^i(x))$,

$M_t(T^{i+1}(x)) = M(T^{i+1}(x))$, $M'(T^{i+1}(x)) = M'_t(T^{i+1}(x))$. Thus, $M(T^{i+1}(x)) \sqsubseteq \ell$. From $M|_\ell = M'|_\ell$, we then have $M'(T^{i+1}(x)) \sqsubseteq \ell$ and $M(T^i(x)) = M'(T^i(x))$. By transitivity, $M_t(T^i(x)) = M'_t(T^i(x))$ and $M'_t(T^{i+1}(x)) \sqsubseteq \ell$. Assume $M_t(bc) \sqsubseteq \ell$. From Lemma 3, we then get that $\mathit{targetAnchor}(C) = \emptyset$. So, $M(bc) = M_t(bc)$ and $M'(bc) = M'_t(bc)$. From $M_t(bc) \sqsubseteq \ell$ and Lemma 7, we get $M(bc) \sqsubseteq \ell$. From $M|_\ell = M'|_\ell$, we then get $M(bc) = M'(bc)$. Thus, by transitivity, we get $M_t(bc) = M'_t(bc)$. And because $M_t(cc) = M'_t(cc)$, we consequently have $M_t|_\ell = M'_t|_\ell$. Thus $c1$ holds.

Induction case:

6.1. $M(\lfloor cc \rceil) \sqsubseteq \ell$ and $M(T(e)) \sqsubseteq \ell$

From $\mathit{mon}(M)$, we then have $M(T^2(e)) \sqsubseteq \ell$. From $M|_\ell = M'|_\ell$ and Lemma 5, we then get $M(T(e)) = M'(T(e))$ and $M(e) = M'(e)$. So, τ and τ' take the same branch. If both take (WL2), then we follow the Base case.

Assume that both take (WL1):
$\tau = \langle \textbf{while } e \textbf{ do } C_1 \textbf{ end}, M\rangle \to \langle C_1; \textbf{while } e \textbf{ do } C_1 \textbf{ end}; \textbf{exit}, M_1\rangle \xrightarrow{*} \langle C_t, M_t\rangle$,
$\tau' = \langle \textbf{while } e \textbf{ do } C_1 \textbf{ end}, M'\rangle \to \langle C_1; \textbf{while } e \textbf{ do } C_1 \textbf{ end}; \textbf{exit}, M'_1\rangle \xrightarrow{*} \langle C'_t, M'_t\rangle$.
We get $M_1(cc) = M'_1(cc)$ from $M(cc) = M'(cc)$ and $M(T(e)) = M'(T(e))$.
We prove $M_1|_\ell = M'_1|_\ell$. Assume $M_1(bc) \sqsubseteq \ell$. Because $M_1(bc) = M(bc)$, we then get $M(bc) \sqsubseteq \ell$. From $M|_\ell = M'|_\ell$, we then get $M(bc) = M'(bc)$. Because $M'_1(bc) = M'(bc)$, we then get $M_1(bc) = M'_1(bc)$. So, $M_1|_\ell = M'_1|_\ell$.
We get $\mathit{mon}(M_1)$ and $\mathit{mon}(M_1)$, from Lemma 2, $\mathit{mon}(M)$, and $\mathit{mon}(M')$.

6.1.1. C_1 terminates normally in the 1st iteration in τ and τ'.
$\tau = \langle \textbf{while } e \textbf{ do } C_1 \textbf{ end}, M\rangle \to \langle C_1; \textbf{while } e \textbf{ do } C_1 \textbf{ end}; \textbf{exit}, M_1\rangle \xrightarrow{*} \langle \textbf{while } e \textbf{ do } C_1 \textbf{ end}; \textbf{exit}, M_2\rangle \xrightarrow{*} \langle C_t, M_t\rangle$,
$\tau' = \langle \textbf{while } e \textbf{ do } C_1 \textbf{ end}, M'\rangle \to \langle C_1; \textbf{while } e \textbf{ do } C_1 \textbf{ end}; \textbf{exit}, M'_1\rangle \xrightarrow{*} \langle \textbf{while } e \textbf{ do } C_1 \textbf{ end}; \textbf{exit}, M'_2\rangle \xrightarrow{*} \langle C'_t, M'_t\rangle$.
Consider:
$\tau_1 = \langle C_1, M_1\rangle \xrightarrow{*} \langle \textbf{stop}, M_2\rangle$
$\tau'_1 = \langle C_1, M'_1\rangle \xrightarrow{*} \langle \textbf{stop}, M'_2\rangle$

Because $M_1(cc) = M_1'(cc)$, $M_1|_\ell = M_1'|_\ell$, $mon(M_1)$, and $mon(M_1)$, we can apply IH[$c1$] on C_1. So, we get $\tau_1|_\ell =_{obs} \tau_1'|_\ell$, $M_2|_\ell = M_2'|_\ell$, and $M_2(cc) = M_2'(cc)$. From $mon(M_1)$, $mon(M_1')$ and Lemma 2, we get $mon(M_2)$, $mon(M_2')$.

Consider traces:

$\tau_2 = \langle \textbf{while } e \textbf{ do } C_1 \textbf{ end}, M_2\rangle \xrightarrow{*} \langle C_3, M_3\rangle$

$\tau_2' = \langle \textbf{while } e \textbf{ do } C_1 \textbf{ end}, M_2'\rangle \xrightarrow{*} \langle C_3', M_3'\rangle$

that terminate (normally or blocked). Because $M_2|_\ell = M_2'|_\ell$, $M_2(cc) = M_2'(cc)$, $mon(M_2)$, and $mon(M_2')$, we can apply IH on the max-number of iterations on τ_2 and τ_2'.

We prove $c2$. Say that C_t is **block**. Then C_3 should be **block**. From IH[$c2$] on τ_2 and τ_2', we then get $\tau_2|_\ell =_{obs} \tau_2'|_\ell$. Because we have $\tau|_\ell =_{obs} \tau_1|_\ell \rightharpoonup \tau_2|_\ell$, $\tau'|_\ell =_{obs} \tau_1'|_\ell \rightharpoonup \tau_2'|_\ell$, $\tau_1|_\ell =_{obs} \tau_1'|_\ell$, and $\tau_2|_\ell =_{obs} \tau_2'|_\ell$, we get $\tau|_\ell =_{obs} \tau'|_\ell$. So, $c2$ holds.

We similarly prove $c3$ and $c4$.

We prove $c1$. Assume τ and τ' terminate normally:

$\tau = \langle \textbf{while } e \textbf{ do } C_1 \textbf{ end}, M\rangle \to \langle C_1; \textbf{while } e \textbf{ do } C_1 \textbf{ end}; \textbf{exit}, M_1\rangle$
$\xrightarrow{*} \langle \textbf{while } e \textbf{ do } C_1 \textbf{ end}; \textbf{exit}, M_2\rangle \xrightarrow{*} \langle \textbf{exit}, M_3\rangle \to \langle \textbf{stop}, M_t\rangle$,

$\tau' = \langle \textbf{while } e \textbf{ do } C_1 \textbf{ end}, M'\rangle \to \langle C_1; \textbf{while } e \textbf{ do } C_1 \textbf{ end}; \textbf{exit}, M_1'\rangle$
$\xrightarrow{*} \langle \textbf{while } e \textbf{ do } C_1 \textbf{ end}; \textbf{exit}, M_2'\rangle \xrightarrow{*} \langle \textbf{exit}, M_3'\rangle \to \langle \textbf{stop}, M_t'\rangle$.

Then τ_2 and τ_2' terminate normally. So, we have

$\tau_2 = \langle \textbf{while } e \textbf{ do } C_1 \textbf{ end}, M_2\rangle \xrightarrow{*} \langle \textbf{stop}, M_3\rangle$

$\tau_2' = \langle \textbf{while } e \textbf{ do } C_1 \textbf{ end}, M_2'\rangle \xrightarrow{*} \langle \textbf{stop}, M_3'\rangle$

By IH[$c1$] on τ_2 and τ_2', we then get $\tau_2|_\ell =_{obs} \tau_2'|_\ell$, $M_3|_\ell = M_3'|_\ell$ and $M_3(cc) = M_3'(cc)$. Because $\tau|_\ell =_{obs} \tau_1|_\ell \rightharpoonup \tau_2|_\ell$, $\tau'|_\ell =_{obs} \tau_1'|_\ell \rightharpoonup \tau_2'|_\ell$, $\tau_1|_\ell =_{obs} \tau_1'|_\ell$, and $\tau_2|_\ell =_{obs} \tau_2'|_\ell$, we get $\tau|_\ell =_{obs} \tau'|_\ell$.

To prove $c1$, we also need to prove that $M_t|_\ell = M_t'|_\ell$ and $M_t(cc) = M_t'(cc)$. Consider:

$\tau_3 = \langle \textbf{exit}, M_3\rangle \to \langle \textbf{stop}, M_t\rangle$

$\tau_3' = \langle \textbf{exit}, M_3'\rangle \to \langle \textbf{stop}, M_t'\rangle$.

From $mon(M_2)$, $mon(M_2')$ and Lemma 2, we get $mon(M_3)$, $mon(M_3')$. Because $M_3|_\ell = M_3'|_\ell$ and $M_3(cc) = M_3'(cc)$, $mon(M_3)$, and $mon(M_3')$, we can use the proof for **exit** (case *7.*) to get $M_t|_\ell = M_t'|_\ell$ and $M_t(cc) = M_t'(cc)$. So, $c1$ holds.

6.1.2. C_1 blocked in both τ and τ' during 1st iteration.
We use IH on C_1.

6.1.3. C_1 blocked in τ, terminates normally in τ'.

$\tau = \langle \textbf{while } e \textbf{ do } C_1 \textbf{ end}, M\rangle \rightarrow \langle C_1; \textbf{while } e \textbf{ do } C_1 \textbf{ end}; \textbf{exit}, M_1\rangle$
$\xrightarrow{*} \langle C_{1t}; \textbf{while } e \textbf{ do } C_1 \textbf{ end}; \textbf{exit}, M_t\rangle$,
$\tau' = \langle \textbf{while } e \textbf{ do } C_1 \textbf{ end}, M'\rangle \rightarrow \langle C_1; \textbf{while } e \textbf{ do } C_1 \textbf{ end}; \textbf{exit}, M'_1\rangle$
$\xrightarrow{*} \langle \textbf{while } e \textbf{ do } C_1 \textbf{ end}; \textbf{exit}, M'_2\rangle \xrightarrow{*} \langle C'_t, M'_t\rangle$.
Consider:
$\tau_1 = \langle C_1, M_1\rangle \xrightarrow{*} \langle C_{1t}, M_t\rangle$
$\tau'_1 = \langle C_1, M'_1\rangle \xrightarrow{*} \langle \textbf{stop}, M'_2\rangle$
$\tau'_2 = \langle \textbf{while } e \textbf{ do } C_1 \textbf{ end}; \textbf{exit}, M'_2\rangle \xrightarrow{*} \langle C'_t, M'_t\rangle$
IH can be applied to τ_1 and τ'_1, because $M_1|_\ell = M'_1|_\ell$, $M_1(cc) = M'_1(cc)$, $mon(M'_1)$, and C_1 is a subcommand of **while** e **do** C_1 **end**.
We invoke (*4.3.*) for τ_1, τ'_1, and τ'_2.

6.2. $M(\lfloor cc\rfloor) \not\sqsubseteq \ell$ or $M(T(e)) \not\sqsubseteq \ell$
Consider:
$\tau_w =$
$\langle \textbf{if } e \textbf{ then } (C_1; \textbf{while } e \textbf{ do } C_1 \textbf{ end}) \textbf{ else skip end}, M\rangle$
$\xrightarrow{*} \langle C_{wt}, M_{wt}\rangle$, and
$\tau_w =$
$\langle \textbf{if } e \textbf{ then } (C_1; \textbf{while } e \textbf{ do } C_1 \textbf{ end}) \textbf{ else skip end}, M'\rangle$
$\xrightarrow{*} \langle C'_{wt}, M'_{wt}\rangle$.
We invoke case *5.2.* for τ_w, τ'_w, and we get:

c1w If C_{wt} and C'_{wt} are both **stop**, then $\tau_w|_\ell =_{obs} \tau'_w|_\ell$, $M_{wt}|_\ell = M'_{wt}|_\ell$, and $M_{wt}(cc) = M'_{wt}(cc)$.

c2w If C_{wt} or C'_{wt} is **block**, then $\tau_w|_\ell =_{obs} \tau'_w|_\ell$.

c3w If C_{wt} is **stop**, C'_{wt} is **block**, and $M'_{wt}(bc) \not\sqsubseteq \ell$, then $M_{wt}(bc) \not\sqsubseteq \ell$.

We prove $c1$. Assume C_t and C'_t are both **stop**. Because $C_t = C_{wt}$ and $C'_t = C'_{wt}$, we have that C_{wt} and C'_{wt} are both **stop**. From **c1w**, we have $\tau_w|_\ell = \tau'_w|_\ell$, $M_{wt}|_\ell = M'_{wt}|_\ell$, and $M_{wt}(cc) = M'_{wt}(cc)$. We have $M_t = M_{wt}$, $M'_t = M'_{wt}$, $\tau|_\ell =_{obs} \tau_w|_\ell$, and $\tau'|_\ell =_{obs} \tau'_w|_\ell$. So, $\tau|_\ell =_{obs} \tau'|_\ell$, $M_t|_\ell = M'_t|_\ell$, and $M_t(cc) = M'_t(cc)$. Thus $c1$ holds.

Similarly, we get $c2$ and $c3$.

$c4$ is trivially true: from $M(\lfloor cc\rfloor) \not\sqsubseteq \ell$ or $M(T(e)) \not\sqsubseteq \ell$ we get $M'_{tp}(\lfloor cc\rfloor) \not\sqsubseteq \ell$.

7. **exit**
We have:

$\tau = \langle \mathbf{exit}, M\rangle \to \langle \mathbf{stop}, M_t\rangle$
$\tau' = \langle \mathbf{exit}, M'\rangle \to \langle \mathbf{stop}, M'_t\rangle$.
$c2$, $c3$, $c4$ are trivially true.
We prove $c1$. We have $\tau|_\ell =_{obs} \epsilon$ and $\tau'|_\ell =_{obs} \epsilon$. So, we need to prove $M_t|_\ell = M_t|_\ell$ and $M_t(cc) = M'_t(cc)$. Because $M(cc) = M'(cc)$, $M_t(cc) = M(cc).pop$, and $M'_t(cc) = M'(cc).pop$, we then get $M_t(cc) = M'_t(cc)$. We now prove $M_t|_\ell = M_t|_\ell$.

7.1. $M_t(\lfloor cc\rfloor) \sqcup M_t(bc) \not\sqsubseteq \ell$ and $M(cc).top.A \neq \emptyset$.
We first prove that $M_t(bc) \not\sqsubseteq \ell$ and $M'_t(bc) \not\sqsubseteq \ell$. Because $M(cc).top.A \neq \emptyset$, we have $M_t(bc) = M(bc) \sqcup M(\lfloor cc\rfloor)$. Because $M_t(\lfloor cc\rfloor) \sqsubseteq M(\lfloor cc\rfloor)$, we get $M_t(\lfloor cc\rfloor) \sqcup M(bc) \sqsubseteq M(\lfloor cc\rfloor) \sqcup M(bc)$, which becomes $M_t(\lfloor cc\rfloor) \sqcup M(bc) \sqcup M(\lfloor cc\rfloor) \sqsubseteq M(\lfloor cc\rfloor) \sqcup M(bc) \sqcup M(\lfloor cc\rfloor)$, which becomes $M_t(\lfloor cc\rfloor) \sqcup M_t(bc) \sqsubseteq M(\lfloor cc\rfloor) \sqcup M(bc)$, due to $M_t(bc) = M(bc) \sqcup M(\lfloor cc\rfloor)$. From $M_t(\lfloor cc\rfloor) \sqcup M_t(bc) \not\sqsubseteq \ell$ we get $M(\lfloor cc\rfloor) \sqcup M(bc) \not\sqsubseteq \ell$. From $M_t(bc) = M(bc) \sqcup M(\lfloor cc\rfloor)$, we then have $M_t(bc) \not\sqsubseteq \ell$. From $M(\lfloor cc\rfloor) \sqcup M(bc) \not\sqsubseteq \ell$ and $M|_\ell = M'|_\ell$, we get $M'(\lfloor cc\rfloor) \sqcup M'(bc) \not\sqsubseteq \ell$. Because $M(cc).top.A \neq \emptyset$ and $M(cc) = M'(cc)$, we have $M'(cc).top.A \neq \emptyset$, too. So, $M'_t(bc) = M'(bc) \sqcup M'(\lfloor cc\rfloor)$. From $M'(\lfloor cc\rfloor) \sqcup M'(bc) \not\sqsubseteq \ell$, we then have $M'_t(bc) \not\sqsubseteq \ell$. So, $M_t(bc) \not\sqsubseteq \ell$ and $M'_t(bc) \not\sqsubseteq \ell$.

Only variables in W change their labels. Let $x \in M(cc).top.W$. Because $M(\lfloor cc\rfloor) \sqcup M(bc) \not\sqsubseteq \ell$, we have $\forall i \geq 1. M_t(T^i(x)) \not\sqsubseteq \ell$. Because $M(cc) = M'(cc)$, we get $x \in M'(cc).top.W$, too. From $M'_t(bc) \not\sqsubseteq \ell$, we have $M'(\lfloor cc\rfloor) \sqcup M'(bc) \not\sqsubseteq \ell$, and thus, $\forall i \geq 1. M'_t(T^i(x)) \not\sqsubseteq \ell$. So, $M_t|_\ell = M'_t|_\ell$. Thus $c1$ holds.

7.2. $M_t(\lfloor cc\rfloor) \sqcup M_t(bc) \not\sqsubseteq \ell$ and $M(cc).top.A = \emptyset$.
Because $M(cc).top.A = \emptyset$, we have $M_t(bc) = M(bc)$. We have $M_t(\lfloor cc\rfloor) \sqsubseteq M(\lfloor cc\rfloor)$. We get $M_t(\lfloor cc\rfloor) \sqcup M(bc) \sqsubseteq M(\lfloor cc\rfloor) \sqcup M(bc)$, which becomes $M_t(\lfloor cc\rfloor) \sqcup M_t(bc) \sqsubseteq M(\lfloor cc\rfloor) \sqcup M(bc)$, due to $M_t(bc) = M(bc)$. From $M_t(\lfloor cc\rfloor) \sqcup M_t(bc) \not\sqsubseteq \ell$, we then have $M(\lfloor cc\rfloor) \sqcup M(bc) \not\sqsubseteq \ell$. Because $M|_\ell = M'|_\ell$, we also get $M'(\lfloor cc\rfloor) \sqcup M'(bc) \not\sqsubseteq \ell$.

We prove that if $M_t(bc) \sqsubseteq \ell$, then $M_t(bc) = M'_t(bc)$. Assume $M_t(bc) \sqsubseteq \ell$. From $M_t(bc) = M(bc)$, we then get $M(bc) \sqsubseteq \ell$. From $M|_\ell = M'|_\ell$, we then get $M(bc) = M'(bc)$. Because $M(cc).top.A = \emptyset$ and $M(cc) = M'(cc)$, we have that $M'(cc).top.A = \emptyset$. So, $M'_t(bc) = M'(bc)$. By transitivity, we then get $M_t(bc) = M'_t(bc)$.

From $M_t(cc) = M'_t(cc)$, $M_t(bc) = M'_t(bc)$, and $M_t(\lfloor cc\rfloor) \sqcup M_t(bc) \not\sqsubseteq \ell$, we then get $M'_t(\lfloor cc \rfloor) \sqcup M'_t(bc) \not\sqsubseteq \ell$.

Only variables in W change their labels. Let $x \in M(cc).top.W$. Because $M(\lfloor cc \rfloor) \sqcup M(bc) \not\sqsubseteq \ell$, we have $\forall i \geq 1. M_t(T^i(x)) \not\sqsubseteq \ell$. Because $M(cc) = M'(cc)$, we get $x \in M'(cc).top.W$, too. Because $M'(\lfloor cc \rfloor) \sqcup M'(bc) \not\sqsubseteq \ell$, we have $\forall i \geq 1. M'_t(T^i(x)) \not\sqsubseteq \ell$. So, $M_t|_\ell = M'_t|_\ell$. Thus $c1$ holds.

7.3. $M_t(\lfloor cc \rfloor) \sqcup M_t(bc) \sqsubseteq \ell$

So, $M_t(bc) \sqsubseteq \ell$. From Lemma 7, we get $M(bc) \sqsubseteq \ell$. From $M|_\ell = M'|_\ell$ and $M(bc) \sqsubseteq \ell$, we also get $M(bc) = M'(bc)$. From $M(cc) = M'(cc)$, we then get $M_t(bc) = M'_t(bc)$.

- Let $M(\lfloor cc \rfloor) \sqcup M(bc) \sqsubseteq \ell$. Let $x \in M(cc).top.W$. Because $M(cc) = M'(cc)$, we get $x \in M'(cc).top.W$. We have:
  $M_t(T^i(x)) \sqsubseteq \ell \Rightarrow M(T^i(x)) \sqsubseteq \ell \Rightarrow M(T^{i-1}(x)) = M'(T^{i-1}(x))$ and $M'(T^i(x)) \sqsubseteq \ell$.
  Because $M(cc) = M'(cc)$ and $M(bc) = M'(bc)$, we then have $M_t(T^{i-1}(x)) = M'_t(T^{i-1}(x))$ and $M'_t(T^i(x)) \sqsubseteq \ell$. Thus, $M_t|_\ell = M'_t|_\ell$. Thus $c1$ holds.
- Let $M(\lfloor cc \rfloor) \sqcup M(bc) \not\sqsubseteq \ell$.
  So, $M'(\lfloor cc \rfloor) \sqcup M'(bc) \not\sqsubseteq \ell$. Let $x \in M(cc).top.W$. Consequently, we have that $\forall i \geq 1$: $M_t(T^i(x)) \not\sqsubseteq \ell$. Because $M(cc) = M'(cc)$, we get $x \in M'(cc).top.W$, and thus $\forall i \geq 1$: $M'_t(T^i(x)) \not\sqsubseteq \ell$. Thus, $M_t|_\ell = M'_t|_\ell$. Thus $c1$ holds.

□

Lemma 2. *Let $\langle C, M \rangle \xrightarrow{*} \langle C', M' \rangle$ be a trace generated by ∞-Enf. If $mon(M)$, then $mon(M')$.*

Proof. We first prove the statement for one-step transition: $\langle C, M \rangle \to \langle C', M' \rangle$, and then we use induction on the number of steps in $\langle C, M \rangle \xrightarrow{*} \langle C', M' \rangle$. We prove the statement for one-step transition, using induction on the rules of ∞-*Enf*'s operational semantics. Assume $mon(M)$. We prove $mon(M')$.

1. (Skip):
Because $M = M'$, we then get $mon(M')$.

2. (AsgnA):
Trivially true, because no label chain is being updated, and thus, $mon(M')$.

3. (AsgnAFail):
   Same arguments as in above case.

4. (AsgnF):
   From $mon(M')$, we have that for every x we have
   $\forall i \geq 1$: $M(T^{i+1}(x)) \sqsubseteq M(T^i(x))$,
   So, we get $\forall i \geq 1$: $M(T^{i+1}(e)) \sqsubseteq M(T^i(e))$. We then have
   $\forall i \geq 1$: $M(T^{i+1}(e)) \sqcup M(\lfloor cc \rfloor) \sqcup M(bc) \sqsubseteq M(T^i(e)) \sqcup M(\lfloor cc \rfloor) \sqcup M(bc)$,
   and thus $\forall i \geq 1$: $M'(T^{i+1}(w)) \sqsubseteq M'(T^i(w))$. So, $mon(M')$.

5. (If1),(If2),(Wl1),(Wl2):
   Trivially true, because no label chain is being updated, and thus, $mon(M')$.

6. (Exit):
   Only label chains of $w \in V$ change. From $mon(M)$, we have for $i \geq 1$, $M(T^{i+1}(w)) \sqsubseteq M(T^i(w))$. Thus, $M(T^{i+1}(w)) \sqcup M(\lfloor cc \rfloor) \sqcup M(bc) \sqsubseteq M(T^i(w)) \sqcup M(\lfloor cc \rfloor) \sqcup M(bc)$. So, $M'(T^{i+1}(w)) \sqsubseteq M'(T^i(w))$. So, $mon(M')$.

7. (Seq1),(Seq2),(SeqF):
   We use the IH.

□

Lemma 3. *If C does not include* **exit** *(or i-***exit***), if $\tau = \langle C, M \rangle \xrightarrow{*} \langle C', M' \rangle$ is generated by ∞-Enf and $M(\lfloor cc \rfloor) \sqcup M(bc) \not\sqsubseteq \ell$, or C is a conditional command (executed under* If, If', *or* Wl *rule—not* IfS *rules) with guard e and $M(T(e)) \not\sqsubseteq \ell$, then*

(i) $\tau|_\ell =_{obs} \epsilon$.

(ii) *if $C' =$* **stop** *and $w \in targetFlex(C)$, then $M'(T^i(w)) \not\sqsubseteq \ell$, for all $T^i(w) \in dom(M')$ where $i \geq 1$.*

(iii) *if $C' =$* **stop** *and $targetAnchor(C) \neq \emptyset$, then $M'(bc) \not\sqsubseteq \ell$.*

Proof. Induction on C (which should not include **exit**).

1. $a := e$
   Assume $M(\lfloor cc \rfloor) \sqcup M(bc) \not\sqsubseteq \ell$.

   If $C' =$ **block**, then $\tau|_\ell = \epsilon$, so (i) holds, and (ii), (iii) are trivially true.

Assume $C' = \textbf{stop}$. So, $M(\lfloor cc \rfloor) \sqcup M(bc) \sqsubseteq M(T(a))$. Because $M(\lfloor cc \rfloor) \sqcup M(bc) \not\sqsubseteq \ell$, we then get $M(T(a)) \not\sqsubseteq \ell$. Thus $\tau|_\ell =_{obs} \epsilon$ and (i) holds.

(ii) is trivially true.

We have $M(\lfloor cc \rfloor) \sqcup M(bc) \sqsubseteq M'(bc)$. Because $M(\lfloor cc \rfloor) \sqcup M(bc) \not\sqsubseteq \ell$, we then get $M'(bc) \not\sqsubseteq \ell$. Thus (iii) holds.

2. $w := e$
   Assume $M(\lfloor cc \rfloor) \sqcup M(bc) \not\sqsubseteq \ell$.

   (iii) is trivially true.

   Because $M(\lfloor cc \rfloor) \sqcup M(bc) \not\sqsubseteq \ell$ and $\forall i \geq 1$: $M(\lfloor cc \rfloor) \sqcup M(bc) \sqsubseteq M'(T^i(w))$, we get $\forall i \geq 1$: $M'(T^i(w)) \not\sqsubseteq \ell$. Thus (ii) holds.

   Also, $\tau|_\ell =_{obs} \epsilon$. Thus (i) holds.

3. $C_1; C_2$
   Assume $M(\lfloor cc \rfloor) \sqcup M(bc) \not\sqsubseteq \ell$.

   - Assume τ involves only the execution of C_1.
     So, τ is blocked, and thus, (ii), (iii) are trivially true.

     We prove (i). Because τ involves only the blocked execution of C_1, consider $\tau_1 = \langle C_1, M \rangle \xrightarrow{*} \langle \textbf{block}, M' \rangle$. From IH on C_1, we get $\tau_1|_\ell =_{obs} \epsilon$. Because $\tau|_\ell = \tau_1|_\ell$, we then get $\tau|_\ell =_{obs} \epsilon$. Thus (i) holds.

   - Assume τ involves execution of C_1 and C_2.
     Then C_1 is executed to normal termination. Consider:
     $\tau_1 = \langle C_1, M \rangle \xrightarrow{*} \langle \textbf{stop}, M_1 \rangle$.
     Also, C_2 might be executed to termination or blocked. Consider:
     $\tau_2 = \langle C_2, M_1 \rangle \xrightarrow{*} \langle C', M' \rangle$.
     From Lemma 8, we get $M(cc) = M_1(cc)$. From Lemma 7, we get $M(bc) \sqsubseteq M_1(bc)$. So, $M(bc) \sqcup M(cc) \sqsubseteq M_1(bc) \sqcup M_1(\lfloor cc \rfloor)$. From $M(\lfloor cc \rfloor) \sqcup M(bc) \not\sqsubseteq \ell$, we then have $M_1(bc) \sqcup M_1(\lfloor cc \rfloor) \not\sqsubseteq \ell$.

     We prove (i). From IH on C_1, we get $\tau_1|_\ell =_{obs} \epsilon$. From IH on C_2, we get $\tau_2|_\ell =_{obs} \epsilon$. Because $\tau|_\ell =_{obs} \tau_1|_\ell \rightharpoonup \tau_2|_\ell$, we get $\tau|_\ell =_{obs} \epsilon$. Thus (i) holds.

     We prove (ii). Assume $C' = \textbf{stop}$ and $w \in \mathit{targetFlex}(C_1; C_2)$. Then $w \in \mathit{targetFlex}(C_1)$ or $w \in \mathit{targetFlex}(C_2)$. Assume $w \in \mathit{targetFlex}(C_2)$. From IH on C_2, we get $\forall i \geq 1$: $M'(T^i(w)) \not\sqsubseteq \ell$. Thus (ii) holds. Assume $w \notin \mathit{targetFlex}(C_2)$. From $w \in \mathit{targetFlex}(C_1; C_2)$, we get $w \in \mathit{targetFlex}(C_1)$. IH on C_1 gives $\forall i \geq 1$: $M_1(T^i(w)) \not\sqsubseteq \ell$. Because $w \notin \mathit{targetFlex}(C_2)$, we have $\forall i \geq 1$: $M_1(T^i(w)) = M'(T^i(w))$. So, $\forall i \geq 1$: $M'(T^i(w)) \not\sqsubseteq \ell$. Thus (ii) holds.

We prove (iii). Assume $C' = \mathbf{stop}$ and $\mathit{targetAnchor}(C_1; C_2) \neq \emptyset$. Then $\mathit{targetAnchor}(C_1) \neq \emptyset$ or $\mathit{targetAnchor}(C_2) \neq \emptyset$. Assume that $\mathit{targetAnchor}(C_2) \neq \emptyset$. From IH on C_2, we get $M'(bc) \not\sqsubseteq \ell$. Thus (iii) holds. Assume $\mathit{targetAnchor}(C_2) = \emptyset$. Because we have that $\mathit{targetAnchor}(C_1; C_2) \neq \emptyset$, we then get $\mathit{targetAnchor}(C_1) \neq \emptyset$. From IH on C_1, we get $M_1(bc) \not\sqsubseteq \ell$. From Lemma 7, we have $M_1(bc) \sqsubseteq M'(bc)$. So, we have $M'(bc) \not\sqsubseteq \ell$. Thus (iii) holds.

4. **if** e **then** C_1 **else** C_2 **end**

Assume $M(\lfloor cc \rfloor) \sqcup M(bc) \not\sqsubseteq \ell$ or $M(T(e)) \not\sqsubseteq \ell$. W.l.o.g. assume that τ involves the execution of C_1.

So:

$\tau = \langle \mathbf{if}\ e\ \mathbf{then}\ C_1\ \mathbf{else}\ C_2\ \mathbf{end}, M\rangle \rightarrow \langle C_1; \mathbf{exit}, M_1\rangle \xrightarrow{*} \langle C', M'\rangle$.

Consider:

$\tau_1 = \langle C_1, M_1\rangle \xrightarrow{*} \langle C_1', M_1'\rangle$.

We have $M(bc) = M_1(bc)$ and $M(\lfloor cc \rfloor) \sqsubseteq M_1(\lfloor cc \rfloor)$. So, $M(\lfloor cc \rfloor) \sqcup M(bc) \sqsubseteq M_1(\lfloor cc \rfloor) \sqcup M_1(bc)$. If $M(\lfloor cc \rfloor) \sqcup M(bc) \not\sqsubseteq \ell$, we then have $M_1(bc) \sqcup M_1(\lfloor cc \rfloor) \not\sqsubseteq \ell$. If $M(T(e)) \not\sqsubseteq \ell$, we then have $M_1(bc) \sqcup M_1(\lfloor cc \rfloor) \not\sqsubseteq \ell$, because $M(T(e)) \sqsubseteq M_1(\lfloor cc \rfloor)$. So, in any case, $M_1(bc) \sqcup M_1(\lfloor cc \rfloor) \not\sqsubseteq \ell$. So, we can apply IH on C_1.

We prove (i). From IH on C_1, we get $\tau_1|_\ell =_{obs} \epsilon$. Because $\tau|_\ell = \tau_1|_\ell$, we have $\tau|_\ell =_{obs} \epsilon$. Thus (i) holds.

We prove (ii). Assume $C' = \mathbf{stop}$ and $w \in \mathit{targetFlex}(C)$. Then $C_1' = \mathbf{stop}$ and $w \in \mathit{targetFlex}(C_1)$ or $w \in \mathit{targetFlex}(C_2)$. We have:

$\tau = \langle \mathbf{if}\ e\ \mathbf{then}\ C_1\ \mathbf{else}\ C_2\ \mathbf{end}, M\rangle \rightarrow \langle C_1; \mathbf{exit}, M_1\rangle \xrightarrow{*} \langle \mathbf{exit}, M_1'\rangle \rightarrow \langle C', M'\rangle$ and

$\tau_1 = \langle C_1, M_1\rangle \xrightarrow{*} \langle \mathbf{stop}, M_1'\rangle$.

Assume $w \in \mathit{targetFlex}(C_1)$. From IH on C_1, we get $\forall i \geq 1$: $M_1'(T^i(w)) \not\sqsubseteq \ell$. Due to the rule for **exit**, we get $\forall i \geq 1$: $M_1'(T^i(w)) \sqsubseteq M'(T^i(w))$. So, $\forall i \geq 1$: $M'(T^i(w)) \not\sqsubseteq \ell$. Thus (ii) holds.

Assume $w \notin \mathit{targetFlex}(C_1)$. From $w \in \mathit{targetFlex}(C)$, we then have $w \in \mathit{targetFlex}(C_2)$. So, $w \in M_1(cc).\mathit{top}.W$. From Lemma 8, we then have $w \in M_1'(cc).\mathit{top}.W$. Due to the rule for **exit**, we get $\forall i \geq 1$: $M_1'(\lfloor cc \rfloor) \sqcup$

$M'_1(bc) \sqsubseteq M'(T^i(w))$. We have $M_1(\lfloor cc \rfloor) \sqcup M_1(bc) \sqsubseteq M'_1(\lfloor cc \rfloor) \sqcup M'_1(bc)$. So, $M'_1(\lfloor cc \rfloor) \sqcup M'_1(bc) \not\sqsubseteq \ell$. Thus, $\forall i \geq 1$: $M'(T^i(w)) \not\sqsubseteq \ell$. Thus (ii) holds.

We prove (iii). Assume $C' =$ **stop** and $\mathit{targetAnchor}(C) \neq \emptyset$ Then $C'_1 =$ **stop** and $\mathit{targetAnchor}(C_1) \neq \emptyset$ or $\mathit{targetAnchor}(C_2) \neq \emptyset$.

Assume $\mathit{targetAnchor}(C_1) \neq \emptyset$. From IH on C_1, we get $M'_1(bc) \not\sqsubseteq \ell$. From Lemma 7, we get $M'_1(bc) \sqsubseteq M'(bc)$. Thus, we have $M'(bc) \not\sqsubseteq \ell$. Thus (iii) holds.

Assume $\mathit{targetAnchor}(C_1) = \emptyset$. From $\mathit{targetAnchor}(C) \neq \emptyset$, we then have $\mathit{targetAnchor}(C_2) \neq \emptyset$. So, $M_1(cc).\mathit{top}.A \neq \emptyset$. From Lemma 8, we then have $M'_1(cc).\mathit{top}.A \neq \emptyset$. Due to the rule for **exit**, we get $M'(bc) = M'_1(\lfloor cc \rfloor) \sqcup M'_1(bc)$. We have $M_1(\lfloor cc \rfloor) \sqcup M_1(bc) \sqsubseteq M'_1(\lfloor cc \rfloor) \sqcup M'_1(bc)$. So, $M'_1(\lfloor cc \rfloor) \sqcup M'_1(bc) \not\sqsubseteq \ell$. Thus, $M'(bc) \not\sqsubseteq \ell$. Thus (iii) holds.

5. **while** e **do** C_1 **end**
   We use induction on the number of iterations executed in τ, and IH on C_1, similar to the above cases.

□

Lemma 4. *If $\tau = \langle C, M\rangle \xrightarrow{*} \langle C', M'\rangle$ is generated by ∞-Enf, and $M(bc) \not\sqsubseteq \ell$, then $\tau|_\ell = \epsilon$.*

Proof. By structural induction on C and Lemma 7. □

Lemma 5. *If $M|_\ell = M'|_\ell$ and $M(T^i(e)) \sqsubseteq \ell$, for $i \geq 1$, then $M(T^{i-1}(e)) = M'(T^{i-1}(e))$.*

Proof. We use structural induction on e.

1. e is n:
   By definition $M(n) = M'(n) = n$. By definition $M(T^i(n)) = M'(T^i(n)) = \bot$, for $i \geq 1$.

2. e is x:
   From $M|_\ell = M'|_\ell$ and $M(T^i(x)) \sqsubseteq \ell$, we have $M(T^{i-1}(x)) = M'(T^{i-1}(x))$.

3. e is $e_1 \oplus e_2$:
   We have $M(T^i(e)) = M(T^i(e_1)) \sqcup M(T^i(e_2))$. From $M(T^i(e)) \sqsubseteq \ell$, we then get $M(T^i(e_1)) \sqsubseteq \ell$ and $M(T^i(e_2)) \sqsubseteq \ell$. By IH on e_1 and e_2, we get $M(T^{i-1}(e_1)) = M'(T^{i-1}(e_1))$ and $M(T^{i-1}(e_2)) = M'(T^{i-1}(e_2))$. We

have $M(e) = M(e_1) \oplus M(e_2) = M'(e_1) \oplus M'(e_2) = M'(e)$. For $i \geq 1$, we have $M(T^i(e)) = M(T^i(e_1)) \sqcup M(T^i(e_2)) = M'(T^i(e_1)) \sqcup M'(T^i(e_2)) = M'(T^i(e))$.

□

Lemma 6. *If $M|_\ell = M'|_\ell$, $mon(M)$, $mon(M')$ and $M(T^i(e)) \not\sqsubseteq \ell$, then $M'(T^i(e)) \not\sqsubseteq \ell$.*

Proof. We prove it by contradiction. Assume $M'(T^i(e)) \sqsubseteq \ell$. From $mon(M)$, we then have $M'(T^{i+1}(e)) \sqsubseteq \ell$. From Lemma 5, we then get $M(T^i(e)) = M'(T^i(e))$. So, $M(T^i(e)) \sqsubseteq \ell$, which is a contradiction. □

Lemma 7. *Let $\langle C, M\rangle \xrightarrow{*} \langle C', M'\rangle$ be a trace generated by ∞-Enf. Then $M(bc) \sqsubseteq M'(bc)$.*

Proof. We first prove that: if $\langle C, M\rangle \rightarrow \langle C', M'\rangle$ is generated by ∞-*Enf*, then $M(bc) \sqsubseteq M'(bc)$. To prove this, we use induction on the rules for ∞-*Enf*'s operational semantics. We then use induction on the number of steps in $\langle C, M\rangle \rightarrow \langle C', M'\rangle$. □

Lemma 8. *Let $\langle C, M\rangle \xrightarrow{*} \langle$**stop**$, M_t\rangle$ be a trace generated by ∞-Enf and let C have no* **exit** *(or i-***exit***), then $M(cc) = M_t(cc)$.*

Proof. We use structural induction on C. □

Lemma 9. *Let $\langle C, M\rangle \xrightarrow{*} \langle C', M'\rangle$ be a trace generated by ∞-Enf. If 2stut(M), then 2stut(M').*

Proof. We first prove the statement for one-step transition:
$\langle C, M\rangle \rightarrow \langle C', M'\rangle$, and then we use induction on the number of steps in $\langle C, M\rangle \xrightarrow{*} \langle C', M'\rangle$. We prove the statement for one-step transition, using induction on the rules of ∞-*Enf*'s operational semantics. Assume, for all x, we have $M(T^2(x)) \sqsubseteq M(T(x))$ and $\forall i > 1$: $M(T^2(x)) = M(T^i(x))$.

1. (Skip):
Because $M = M'$, we then get $M'(T^2(x)) \sqsubseteq M'(T(x))$ and
$\forall i > 1$: $M'(T^2(x)) = M'(T^i(x))$.

2. (AsgnA):
Trivially true, because no label chain is being updated, and thus, we have
$\forall i \geq 1$: $M(T^i(x)) = M'(T^i(x))$.

3. (AsgnAFail):
   Same arguments as in above case.

4. (AsgnF):
   Because for every x we have $M(T^2(x)) \sqsubseteq M(T(x))$, we get $M(T^2(e)) \sqsubseteq M(T(e))$. We then have
   $M(T^2(e)) \sqcup M(\lfloor cc \rfloor) \sqcup M(bc) \sqsubseteq M(T(e)) \sqcup M(\lfloor cc \rfloor) \sqcup M(bc)$,
   and thus $M'(T^2(w)) \sqsubseteq M'(T(w))$.
   Similarly, we get $\forall i > 1$: $M'(T^2(w)) = M'(T^i(w))$.

5. (If1),(If2),(Wl1),(Wl2):
   Trivially true, because no label chain is being updated, and thus,
   $\forall i \geq 1$: $M(T^i(x)) = M'(T^i(x))$.

6. (Exit):
   Only label chains of $w \in W$ change. We have $M'(T^2(w)) = M(T^2(w)) \sqcup M(\lfloor cc \rfloor) \sqcup M(bc)$ and $M'(T(w) = M(T(w)) \sqcup M(\lfloor cc \rfloor) \sqcup M(bc)$. Because $M(T^2(w)) \sqsubseteq M(T(w))$, we then get $M'(T^2(w)) \sqsubseteq M'(T(w))$. Similarly, we get $\forall i > 1$: $M'(T^2(w)) = M'(T^i(w))$.

7. (Seq1),(Seq2),(SeqF):
   We use IH.

$\square$

Theorem 2. *k-Enf is an enforcer on R for k-BNI($\mathcal{L}$), for any $\mathcal{L}$ and $k \geq 2$.*

Proof. It is easy to prove that k-*Enf* is an enforcer on R and satisfies restrictions (E1), (E2), and (E3) by induction on the rules for k-*Enf*. We omit the details.

We now prove k-BNI(k-*Enf*$, \mathcal{L}, C)$, for a command C, a lattice $\mathcal{L}$, and $k \geq 2$. Consider $\ell \in \mathcal{L}$. Take M, M' with $M \models \mathcal{H}_0(k\text{-}\mathit{Enf}, \mathcal{L}, C)$, $M' \models \mathcal{H}_0(k\text{-}\mathit{Enf}, \mathcal{L}, C)$, $M|_\ell = M'|_\ell$, and finite traces $\tau = \mathit{trace}_{k\text{-}\mathit{Enf}}(C, M)$ and $\tau' = \mathit{trace}_{k\text{-}\mathit{Enf}}(C, M')$, where $\tau = \langle C, M\rangle \xrightarrow{*} \langle C_t, M_t\rangle$, and $\tau' = \langle C, M'\rangle \xrightarrow{*} \langle C'_t, M'_t\rangle$.

We prove $\tau|^k_\ell =_{\mathit{obs}} \tau'|^k_\ell$ or equivalently $\tau|_\ell =_{\mathit{obs}} \tau'|_\ell$, because k-*Enf* generates observation up to kth tag. From $M \models \mathcal{H}_0(k\text{-}\mathit{Enf}, \mathcal{L}, C)$ and $M' \models \mathcal{H}_0(k\text{-}\mathit{Enf}, \mathcal{L}, C)$, we get $M(cc) = M'(cc)$, $\mathit{mon}(M)$, and $\mathit{mon}(M')$. There exists M_I such that $M_I \models \mathcal{H}_0(\infty\text{-}\mathit{Enf}, \mathcal{L}, C)$, $M_I =_k M$, and $\mathit{kstut}(M_I)$.

Similarly, there exists M'_I such that $M'_I \models \mathcal{H}_0(\infty\text{-}Enf, \mathcal{L}, C)$, $M'_I =_k M'$, and $kstut(M'_I)$.

We prove $M_I|_\ell = M'_I|_\ell$. Due to $M|_\ell = M'|_\ell$, $M_I =_k M$, and $M'_I =_k M'$, it suffices to examine $\forall x \colon \forall i > k \colon T^i(x)$. Assume $M_I(T^i(x)) \sqsubseteq \ell$. From $kstut(M_I)$, we then get $M_I(T^k(x)) \sqsubseteq \ell$. From $M_I =_k M$, we then have $M(T^k(x)) \sqsubseteq \ell$. By definition of k-Enf, we have $T^{k+1}(x) = T^k(x)$, and thus $M(T^{k+1}(x)) \sqsubseteq \ell$. From $M|_\ell = M'|_\ell$, we then get $M(T^k(x)) = M'(T^k(x))$. From $M_I =_k M$, $kstut(M_I)$, $M'_I =_k M'$, and $kstut(M'_I)$, we have $M_I(T^{i-1}(x)) = M'_I(T^{i-1}(x))$ and $M_I(T^i(x)) = M'_I(T^i(x))$. So, $M_I|_\ell = M'_I|_\ell$.

We have $M_I(cc) = M'_I(cc)$, due to $M(cc) = M'(cc)$, $M_I =_k M$, and $M'_I =_k M'$. We have $mon(M_I)$, due to $mon(M)$, $M_I =_k M$, and $kstut(M_I)$. Similarly, we have $mon(M'_I)$.

Consider
$\tau_I = trace_{\infty\text{-}Enf}(C, M_I) = \langle C, M_I\rangle \xrightarrow{*} \langle C_t, M_{It}\rangle$, and
$\tau'_I = trace_{\infty\text{-}Enf}(C, M'_I) = \langle C, M'_I\rangle \xrightarrow{*} \langle C'_t, M'_{It}\rangle$.
From Lemma 10, and applying induction on the number of steps in τ and τ', we get $\tau_I|^k_\ell = \tau|_\ell$ and $\tau'_I|^k_\ell = \tau'|_\ell$. Because ∞-*Enf* satisfies BNI+ (Lemma 1), we get $\tau_I|_\ell =_{obs} \tau'_I|_\ell$. We then get $\tau_I|^k_\ell =_{obs} \tau'_I|^k_\ell$. So, $\tau|_\ell =_{obs} \tau'|_\ell$. ☐

Lemma 10. *Consider $\tau = \langle C, M\rangle \to \langle C_n, M_n\rangle$ be a step under k-Enf with $k \geq 2$. Let $M' =_k M$, $kstut(M')$, and $\tau' = \langle C, M'\rangle \to \langle C'_n, M'_n\rangle$ be a step under ∞-Enf. Then, $M'_n =_k M_n$, $kstut(M'_n)$, $C_n = C'_n$, and $\tau|_\ell = \tau'|^k_\ell$.*

Proof. Structural induction on C.

1. C is **skip**:
   We have $M = M_n$ and $M' = M'_n$. So, $M'_n =_k M_n$ and $kstut(M'_n)$. Also, $C_n = C'_n =$ **stop** and $\tau|_\ell = \tau'|^k_\ell = \epsilon$.

2. C is $a := e$:
   $G_{a:=e}$ is $T(e) \sqcup M(\lfloor cc \rfloor) \sqcup bc \sqsubseteq T(a)$. Because $M' =_k M$ and $k \geq 2$, we have $M(e) = M'(e)$, $M(T(e)) = M'(T(e))$, $M(T^2(e)) = M'(T^2(e))$, $M(T(a)) = M'(T(a))$, $M(cc) = M'(cc)$, $M(bc) = M'(bc)$. So, τ and τ' both are generated from AsgnA, or both from AsgnAFail. So, $C_n = C'_n$. Also, $M_n(bc) = M'_n(bc)$. Thus, $M'_n =_k M_n$. Because no label chain changed, $kstut(M')$ gives $kstut(M'_n)$. Because $M(e) = M'(e)$, we also get $\tau|_\ell = \tau'|^k_\ell$.

3. C is $w := e$:
   We have $C_n = C'_n =$ **stop**. We have $\forall 0 \leq i \leq k$:

$M'_n(T^i(w)) = M'(T^i(e)) \sqcup M'(cc) \sqcup M'(bc) =$
$M(T^i(e)) \sqcup M(cc) \sqcup M(bc) = M_n(T^i(w))$.
Also, $\forall i > k$, we have $M'_n(T^i(w)) = M'_n(T^k(w))$. So, $M'_n =_k M_n$ and *kstut*(M'_n). We have $\tau|_\ell = \tau'|^k_\ell$, because $\forall 0 \leq i \leq k+1$ we have $M_n(T^i(w)) = M'_n(T^i(w))$.

4. C is **if** e **then** C_1 **else** C_2 **end**:
From $M' =_k M$, we get $M(e) = M'(e)$, $M(T(e)) = M'(T(e))$, $M(cc) = M'(cc)$, $M(bc) = M'(bc)$. Both τ and τ' take the same branch. Say C_1:
$\tau = \langle C, M\rangle \to \langle C_1; \textbf{exit}, M_1\rangle$
$\tau' = \langle C, M'\rangle \to \langle C_1; \textbf{exit}, M'_1\rangle$.
We have $M_1(cc) = M'_1(cc)$. So, $M'_1 =_k M_1$. Because no label chain changed, *kstut*(M') gives *kstut*(M'_1). Also, $\tau|_\ell = \tau'|_\ell = \epsilon$ and $C_n = C'_n = C_1; \textbf{exit}$.

5. C is **while** e **do** C_1 **end**:
Similarly to above.

6. C is $C_1; C_2$:
By IH on C_1.

7. C is **exit**:
We have $C_n = C'_n = \textbf{stop}$. Also $\tau|_\ell = \tau'|_\ell = \epsilon$. From $M' =_k M$, we get $M(cc) = M'(cc)$ and $M(bc) = M'(bc)$. So, $M_t(cc) = M'_t(cc)$ and $M_t(bc) = M'_t(bc)$. Only variables in W change. Assume $w \in W$. We have that
$\forall 0 \leq i \leq k$: $M'_n(T^i(w)) = M'(T^i(w)) \sqcup M'(cc) \sqcup M'(bc) =$
$M(T^i(w)) \sqcup M(cc) \sqcup M(bc) = M_n(T^i(w))$.
Alos, $\forall i > k$, we have $M'_n(T^i(w)) = M'_n(T^k(w))$. So, $M'_n =_k M_n$ and *kstut*(M'_n).

□

Lemma 11. *If $\langle C, M\rangle \to \langle C', M'\rangle$ according to k-Enf, and 2stut(M), then 2stut(M').*

Proof. Assume $\langle C, M_I\rangle \to \langle C'_I, M'_I\rangle$ according to ∞-*Enf*, where $M_I =_k M$ and *kstut*(M_I). From Lemma 10, we get $M'_I =_k M'$ and *kstut*(M'_I). Because *2stut*(M), $M_I =_k M$, and *kstut*(M_I), we have *2stut*(M_I). From Lemma 9, we get *2stut*(M'_I). From $M'_I =_k M'$, we then get *2stut*(M'). □

$$
\text{(IFS1)}\ \frac{\begin{array}{c}\exists i\colon 1 \leq i \leq k\colon \mathit{isSimple}(\textbf{if } e \textbf{ then } C_1 \textbf{ else } C_2 \textbf{ end}, M, i)\\ M(e) \neq 0 \qquad cc' = M(cc).\mathit{push}(\langle M(T(e)),\ \emptyset,\ \emptyset\rangle\end{array}}{\langle \textbf{if } e \textbf{ then } C_1 \textbf{ else } C_2 \textbf{ end}, M\rangle \rightarrow \qquad \langle C_1; i\textbf{-exit}, M[cc \mapsto cc']\rangle}
$$

$$
\text{(IFS2)}\ \frac{\begin{array}{c}\exists i\colon 1 \leq i \leq k\colon \mathit{isSimple}(\textbf{if } e \textbf{ then } C_1 \textbf{ else } C_2 \textbf{ end}, M, i)\\ M(e) = 0 \qquad cc' = M(cc).\mathit{push}(\langle M(T(e)),\ \emptyset,\ \emptyset\rangle\end{array}}{\langle \textbf{if } e \textbf{ then } C_1 \textbf{ else } C_2 \textbf{ end}, M\rangle \rightarrow \qquad \langle C_2; i\textbf{-exit}, M[cc \mapsto cc']\rangle}
$$

$$
\text{(EXIT\_IFS)}\ \frac{cc' = cc.\mathit{pop}}{\langle i\textbf{-exit}, M\rangle \rightarrow \qquad \langle \textbf{stop}, M[\forall j\colon i < j \leq k\colon T^j(w_i) \mapsto \bot,\ cc \mapsto cc']\rangle}
$$

Figure 10: Rules for simple **if** command

C Optimized Enforcer *k-Eopt*

We sketch the construction of *k-Eopt*. We add two rules for **if** command (one for each truth value of the guard) to *k-Enf*. These new rules apply to a simple **if** command. We add a premise to the existing rules for **if** command, so that these rules are triggered when this **if** command is not simple. The new rules for simple **if** command augment the taken branch with a new delimiter i-**end**, and we add one rule for i-**end** to *k-Enf*; this rule sets certain labels of label chains to $\bot$. Notice, there are programs where *k-Eopt* produces more permissive label chains than those produced by *k-Enf*.

Figure 10 gives the rules for augmenting *k-Enf* in order to obtain *k-Eopt*. Function $\mathit{isSimple}(C, M, i)$ decides whether a command C is simple:

(i) C is of the form **if** $a > 0$ **then** $w_i := e$ **else** $w_i := n$ **end**,

(ii) a is an anchor variable,

(iii) w_i is a flexible variable,

(iv) $i = 1$ and $M(T(e)) = \bot$, or
$i > 1$, $M(T^{i-1}(e)) \neq \bot$, and $M(T^i(e)) = \bot$,

(v) n is a constant,

(vi) C is context-free (e.g., $M(cc) = \epsilon$ and $M(bc) = \bot$).

Notice that if $isSimple(C, M, i)$ holds, then $isSimple(C, M, j)$ does not hold for $j \neq i$, due to (iv) and monotonically decreasing label chains.

As an example, we show how *k-Eopt* deduces label chains for the following simple **if**:

$$\textbf{if } m > 0 \textbf{ then } w := h \textbf{ else } w := 4 \textbf{ end} \tag{21}$$

where anchor variable m is associated with $\langle \mathsf{M}, \perp, \perp, \perp \rangle$, anchor variable h is associated with $\langle \mathsf{H}, \perp, \perp, \perp \rangle$, and $h \neq 4$. Without considering the context (i.e., $m > 0$), flexible variable w would be associated with either $\langle \mathsf{H}, \perp, \perp, \perp \rangle$ (due to $w := h$) or $\langle \perp, \perp, \perp, \perp \rangle$ (due to $w := 4$), when execution of assignments ends. Here, only w and $T(w)$ reveal information about guard $m > 0$. So, at the end of the conditional command, only $T(w)$ and $T^2(w)$ should be updated with the sensitivity of the context $T(m) = \mathsf{M}$. Thus, if $m > 0$, then w is associated with $\langle \mathsf{H}, \mathsf{M}, \perp, \perp \rangle$, at the end of the conditional command. Otherwise, w is associated with $\langle \mathsf{M}, \mathsf{M}, \perp, \perp \rangle$. Notice that, in both cases, the meta-meta label of w is strictly less restrictive than its metalabel. So, using the metalabel to specify its own sensitivity would be conservative. In particular, using rules from *k-Enf*, w would be associated with $\langle \mathsf{M}, \mathsf{M}, \mathsf{M}, \mathsf{M} \rangle$ or $\langle \mathsf{H}, \mathsf{M}, \mathsf{M}, \mathsf{M} \rangle$ at the end of the execution. Consequently, *k-Enf* deduces less permissive label chains than *k-Eopt*.

Consider now how *k-Eopt* produces label chains for the following simple **if**:

$$\textbf{if } a > 0 \textbf{ then } w_i := e \textbf{ else } w_i := n \textbf{ end}$$

where $T(a) = A$, $T^j(e) \neq \perp$ for $j < i$, and $T^j(e) = \perp$ for $j \geq i$. Without considering the context, we have

$$\forall j \geq i \colon T^j(w_i) = \perp$$

at the end of both branches. Only $T^j(w_i)$, for $j < i$, reveal information about guard $a > 0$. So, at the end of the conditional command, only $T^{j+1}(w_i)$, for $j < i$, should be updated with $T(a) = A$. Thus, at the end of the conditional command, we always have

$$\forall j > i \colon T^j(w_i) = \perp.$$

So, when execution exits a simple **if** command, $T^j(w_i)$ can be set to $\perp$, for every $j > i$.

Consider now lattice $\mathcal{L}_3 \triangleq \langle\{\mathsf{H},\mathsf{M},\mathsf{L}\},\sqsubseteq\rangle$ with $\bot = \mathsf{L} \sqsubset \mathsf{M} \sqsubset \mathsf{H}$ and the following program:

$$\begin{aligned}&\textbf{if } m > 0 \textbf{ then } w := h \textbf{ else } w := 4 \textbf{ end};\\&\textbf{if } l > 0 \textbf{ then } w' := w \textbf{ else } w := m \textbf{ end};\\&w'' := w'\end{aligned}$$

where l is anchor variable with $T(l) = \bot$ and w', w'' are flexible variables. If $m \not> 0$ and $l > 0$, then w'' is associated with $\langle\mathsf{M},\mathsf{M},\bot,\bot\rangle$. If $l \not> 0$, then w'' is associated with $\langle\mathsf{M},\bot,\bot,\bot\rangle$. So, *k-Eopt* produces 2-precise 2-varying label chains for the target variable w''. Such an example can be extended to show that *k-Eopt* can produce k-precise k-varying label chains.

For enforcer *k-Eopt*, we have $n_{k\text{-}Eopt} = k + 1$, $Aux_{k\text{-}Eopt} = \{cc, bc\}$, $Init_{k\text{-}Eopt}(cc) = \epsilon$, and $Init_{k\text{-}Eopt}(bc) = \bot$.

D Soundness of *k-Eopt*

Lemma 12. *k-Eopt is an enforcer on R for k-BNI with $k \geq 2$.*

Proof. We first add the rules in Figure 10 to *k-Enf* and retrieve *k-Eopt*. Also, we substitute (If1) and (If2) with:

$$(\text{If1'})\ \frac{\begin{array}{c}\nexists i\colon 1 \leq i \leq k\colon \mathit{isSimple}(\textbf{if } e \textbf{ then } C_1 \textbf{ else } C_2 \textbf{ end}, M, i)\\ M(e) \neq 0 \qquad W = \mathit{targetFlex}(C_2)\\ A = \mathit{targetAnchor}(C_2) \qquad cc' = M(cc).\mathit{push}(\langle M(T(e)),\ W,\ A\rangle)\end{array}}{\langle \textbf{if } e \textbf{ then } C_1 \textbf{ else } C_2 \textbf{ end}, M\rangle \to \qquad \langle C_1; \textbf{exit}, M[cc \mapsto cc']\rangle}$$

$$(\text{If2'})\ \frac{\begin{array}{c}\nexists i\colon 1 \leq i \leq k\colon \mathit{isSimple}(\textbf{if } e \textbf{ then } C_1 \textbf{ else } C_2 \textbf{ end}, M, i)\\ M(e) = 0 \qquad W = \mathit{targetFlex}(C_1)\\ A = \mathit{targetAnchor}(C_1) \qquad cc' = M(cc).\mathit{push}(\langle M(T(e)),\ W,\ A\rangle)\end{array}}{\langle \textbf{if } e \textbf{ then } C_1 \textbf{ else } C_2 \textbf{ end}, M\rangle \to \qquad \langle C_2; \textbf{exit}, M[cc \mapsto cc']\rangle}$$

It is easy to prove that *k-Eopt* is an enforcer on R and satisfies restrictions (E1), (E2), and (E3) by induction on the rules for *k-Eopt*.[9] We omit the details.

[9]For *k-Eopt*, we could use **exit** and introduce a new auxiliary for tracking when a simple **if** is executed. For simplicity, we instead introduce a new conditional delimiter i-**exit** and extend definition $\langle C, M\rangle =^0 \langle C', M'\rangle$ to hold even if the syntax of conditional delimiters that appear in C and C' is different.

We prove that BNI+(k-*Eopt*,$\mathcal{L}$,C) holds, for a command C and a lattice $\mathcal{L}$.
Assume $\ell \in \mathcal{L}$ and

$$M|_\ell = M'|_\ell,$$
$$M(cc) = M'(cc), mon(M), mon(M')$$
$$\tau = \langle C, M\rangle \xrightarrow{*} \langle C_t, M_t\rangle \text{ according to } k\text{-}Eopt,$$
$$\tau' = \langle C, M'\rangle \xrightarrow{*} \langle C'_t, M'_t\rangle \text{ according to } k\text{-}Eopt$$

where C_t and C'_t are terminations (normal or blocked).

We prove $c1$, $c2$, $c3$, and $c4$. We use structural induction on C. We build on the proof of Lemma 1. That proof uses lemmata 2, 3, 4, 5, 6, 7, and 8, which all still hold for k-*Eopt*.
If C is **skip** or $a := e$, then k-*Eopt* and ∞-*Enf* use the same rules. So, we follow the same proof as in Case 1 and Case 2 of Lemma 1.
If C is $w := e$, then k-*Eopt* and ∞-*Enf* use the same rule up to the kth tag. So, we follow the same proof as in Case 3 of Lemma 1 by bounding $r \leq k$ and $i \leq k$ and recalling $T^k(x) = T^{k+1}(x)$ (dy definition of k-*Eopt*).
If C is $C_1; C_2$, **while** e **do** C_t **end**, or **exit**, then k-*Eopt* and ∞-*Enf* use the same rules up to the kth tag. We follow the same proof as in Cases 4,6,7 of Lemma 1 by bounding $i \leq k$ and recalling $T^k(x) = T^{k+1}(x)$ (dy definition of k-*Eopt*).

Now, it suffices to prove that BNI+(k-*Eopt*,$\mathcal{L}$,C) holds, where C is an **if**. We first prove that if *isSimple*(C, M, i) holds for some $1 \leq i \leq k$, then *isSimple*(C, M', i) holds, too. Because *isSimple*(C, M, i) holds, we get:

$$C \text{ is of the form } \textbf{if } a > 0 \textbf{ then } w_i := e_i \textbf{ else } w_i := n \textbf{ end}, \tag{22}$$
$$a \text{ is an anchor variable,} \tag{23}$$
$$w_i \text{ is a flexible variable,} \tag{24}$$
$$\begin{aligned}&i = 1 \text{ and } M(T(e_i)) = \bot, \text{ or}\\ &i > 1 \text{ and } M(T^{i-1}(e_i)) \neq \bot \text{ and } M(T^i(e_i)) = \bot\end{aligned} \tag{25}$$
$$n \text{ is a constant,} \tag{26}$$
$$C \text{ is context-free (e.g., } M(cc) = \epsilon \text{ and } M(bc) = \bot). \tag{27}$$

- From (25), $mon(M)$, and $M|_\ell = M'|_\ell$, we have $M(T^i(e_i)) = M'(T^i(e_i))$ and if $i > 1$, then $M(T^{i-1}(e_i)) = M'(T^{i-1}(e_i))$.

So,

$$\begin{aligned}&i = 1 \text{ and } M'(T(e_i)) = \bot, \text{ or}\\&i > 1 \text{ and } M'(T^{i-1}(e_i)) \neq \bot \text{ and } M'(T^i(e_i)) = \bot\end{aligned} \tag{28}$$

- From (27), we have $M(cc) = \epsilon$. From $M(cc) = M'(cc)$, we then get

$$M'(cc) = \epsilon. \tag{29}$$

- From (27), we have $M(bc) = \bot$. From $M|_\ell = M'|_\ell$, we then get

$$M'(bc) = \bot. \tag{30}$$

From (22), (23), (24), (28), (26), (29), and (30), we get that $\mathit{isSimple}(C, M', i)$ holds. So, if $\mathit{isSimple}(C, M, i)$ holds, then $\mathit{isSimple}(C, M', i)$ holds, too. Similarly, if $\mathit{isSimple}(C, M', i)$ holds, then $\mathit{isSimple}(C, M, i)$ holds. Thus, τ and τ' both use IfS or If'. If both τ and τ' use If' (i.e, If1' or If2'), then we follow Case 5 of Lemma 1.

So, it remains to handle the case where τ and τ' both use IfS. A simple **if** does not contain assignments to anchor variables. So, a trace of a simple **if** never stops before normal termination. Thus, $c2$, $c3$, and $c4$ are trivially true.

We prove $c1$. Assume C is **if** $a > 0$ **then** $w_i := e_i$ **else** $w_i := n$ **end** and

$$\begin{aligned}&M|_\ell = M'|_\ell,\\&M(cc) = M'(cc), \mathit{mon}(M), \mathit{mon}(M')\\&\tau = \langle C, M\rangle \to \langle C_b; i\text{-}\mathbf{exit}, M_b\rangle \to \langle i\text{-}\mathbf{exit}, M_e\rangle \to \langle \mathbf{stop}, M_t\rangle,\\&\tau' = \langle C, M'\rangle \to \langle C'_b; i\text{-}\mathbf{exit}, M'_b\rangle \to \langle i\text{-}\mathbf{exit}, M'_e\rangle \to \langle \mathbf{stop}, M'_t\rangle\end{aligned}$$

where C_b and C'_b are either $w_i := e_i$ or $w_i := n$.

We prove $\tau|_\ell =_{obs} \tau'|_\ell$, $M_t|_\ell = M'_t|_\ell$, and $M_t(cc) = M'_t(cc)$, in the case τ and τ' both use IfS (i.e, IfS1 or IfS2). From $M(cc) = M'(cc)$, IfS, and Exit_IfS, we get $M_t(cc) = M'_t(cc)$. It remains to prove that $\tau|_\ell =_{obs} \tau'|_\ell$ and $M_t|_\ell = M'_t|_\ell$.

We first compute the possible label chains that w_i may be associated with at different points of the execution of C. Notice that by the definition of simple **if** we have $\lfloor cc\rfloor = \bot$ and $bc = \bot$ at the beginning of its execution.

(I) After execution of $w_i := e_i$:
From AsgnF, IfS, and (25) we have:
$\forall j$: $1 \leq j < i$: $T^j(w_i) = T^j(e_i) \sqcup T(a)$ and
$\forall j$: $i \leq j \leq k$: $T^j(w_i) = T(a)$.

(II) After execution of $w_i := n$:
From AsgnF and IfS we have:
$\forall j$: $1 \leq j \leq k$: $T^j(w_i) = T(a)$.

(III) After execution of i-**exit** when $a > 0$ holds:
From Exit_IfS and (I) we have:
$\forall j$: $1 \leq j < i$: $T^j(w_i) = T^j(e_i) \sqcup T(a)$ and $T^i(w_i) = T(a)$ and
$\forall j$: $i < j \leq k$: $T^j(w_i) = \bot$.

(IV) After execution of i-**exit** when $a \not> 0$ holds:
From Exit_IfS and (II) we have:
$\forall j$: $1 \leq j \leq i$: $T^j(w_i) = T(a)$ and $\forall j$: $i < j \leq k$: $T^j(w_i) = \bot$.

By definition of anchor variables, $M(T^2(a)) = \bot$. From $M|_\ell = M'|_\ell$, we then get $M(T(a)) = M'(T(a))$.

We prove $\tau|_\ell =_{obs} \tau'|_\ell$ and $M_t|_\ell = M'_t|_\ell$.

1. $M(T(a)) \sqsubseteq \ell$:
From $M|_\ell = M'|_\ell$, we get $M(a) = M'(a)$. So, τ and τ' take the same branch. We first prove $\tau|_\ell =_{obs} \tau'|_\ell$. Because these observations might involve only w_i and its associated label chain, it suffices to show that: for j such that $1 \leq j \leq k+1$, if $M_e(T^j(w_i)) \sqsubseteq \ell$, then $M'_e(T^j(w_i)) \sqsubseteq \ell$ and $M_e(T^{j-1}(w_i)) = M'_e(T^{j-1}(w_i))$. We examine two cases based on the branch that is executed.

- Branch $w_i := e_i$ is executed.
  Consider j such that $i+1 \leq j \leq k+1$.
  Due to (I), we have $M_e(T^{j-1}(w_i)) = M'_e(T^{j-1}(w_i)) = M(T(a))$. Because $T^{k+1}(w_i) = T^k(w_i)$, we also have that
  $M_e(T^{k+1}(w_i)) = M'_e(T^{k+1}(w_i)) = M(T(a))$.
  Consider $j = i$. $M_e(T^j(w_i)) \sqsubseteq \ell$ and $M'_e(T^j(w_i)) \sqsubseteq \ell$ hold because from (I), we have $M_e(T^j(w_i)) = M'_e(T^j(w_i)) = M(T(a))$ and $M(T(a)) \sqsubseteq \ell$. From (25) and $M|_\ell = M'|_\ell$ we have $M(T^{i-1}(e_i)) = M'(T^{i-1}(e_i))$, and thus, (I) gives $M_e(T^{j-1}(w_i)) = M(T^{i-1}(e_i)) \sqcup M(T(a)) = M'(T^{i-1}(e_i)) \sqcup M'(T(a)) = M'_e(T^{j-1}(w_i))$.

Consider $j < i$. Assume $M_e(T^j(w_i)) \sqsubseteq \ell$. Then, from (I), we have $M(T^j(e_i)) \sqsubseteq \ell$. From $M|_\ell = M'|_\ell$ and Lemma 5, we then have $M(T^{j-1}(e_i)) = M'(T^{j-1}(e_i))$. For $j = 1$, we then have $M(e_i) = M'(e_i)$. For $j \neq 1$, (I) gives $M_e(T^{j-1}(w_i)) = M(T^{j-1}(e_i)) \sqcup M(T(a)) = M'(T^{j-1}(e_i)) \sqcup M'(T(a)) = M'_e(T^{j-1}(w_i))$.

- Branch $w_i := n$ is executed.
  From (II) and because $T^{k+1}(w_i) = T^k(w_i)$, we get
  $\forall j$: $1 \leq j \leq k+1$: $M_e(T^j(w_i)) = M'_e(T^j(w_i)) = M(T(a))$.
  Also, $M_e(w_i) = M'_e(w_i) = n$.

So, $\tau|_\ell =_{obs} \tau'|_\ell$ holds.

We now prove $M_t|_\ell = M'_t|_\ell$. Because $M_t(cc) = M'_t(cc)$ and because bc is not modified, it suffices to prove $M_t|_\ell = M'_t|_\ell$ for the label chain of w_i. We prove for j with $1 \leq j \leq k{+}1$ that if $M_t(T^j(w_i)) \sqsubseteq \ell$, then $M'_t(T^j(w_i)) \sqsubseteq \ell$ and $M_t(T^{j-1}(w_i)) = M'_t(T^{j-1}(w_i))$. We examine two cases based on the branch that is executed.

- Branch $w_i := e_i$ is executed.
  From (III) and because $T^{k+1}(w_i) = T^k(w_i)$, we get
  $\forall j$: $i \leq j \leq k+1$: $M_t(T^j(w_i)) = M'_t(T^j(w_i))$.
  So, it suffices to examine $j \leq i$.

  Consider $j = i$. From (25) and $M|_\ell = M'|_\ell$ we have $M(T^{i-1}(e_i)) = M'(T^{i-1}(e_i))$, and thus, (III) gives $M_t(T^{j-1}(w_i)) = M(T^{i-1}(e_i)) \sqcup M(T(a)) = M'(T^{i-1}(e_i)) \sqcup M(T(a)) = M'_t(T^{j-1}(w_i))$.

  Consider $j < i$. Assume $M_t(T^j(w_i)) \sqsubseteq \ell$. Then, from (III), we have $M(T^j(e_i)) \sqsubseteq \ell$. From $M|_\ell = M'|_\ell$, we then have $M(T^{j-1}(e_i)) = M'(T^{j-1}(e_i))$. For $j = 1$, we then have $M(e_i) = M'(e_i)$. For $j \neq 1$, we then have $M_t(T^{j-1}(w_i)) = M(T^{j-1}(e_i)) \sqcup M(T(a)) = M'(T^{j-1}(e_i)) \sqcup M'(T(a)) = M'_t(T^{j-1}(w_i))$.

- Branch $w_i := n$ is executed.
  From (IV) and because $T^{k+1}(w_i) = T^k(w_i)$, we get
  $\forall j$: $1 \leq j \leq k+1$: $M_t(T^j(w_i)) = M'_t(T^j(w_i))$.
  Also, $M_t(w_i) = M'_t(w_i) = n$.

Thus, $M_t|_\ell = M'_t|_\ell$ holds.

2. $M(T(a)) \not\sqsubseteq \ell$:
Traces τ and τ' may take different branches. From (I), (II), $M(T(a)) \not\sqsubseteq \ell$,

and because $T^{k+1}(w_i) = T^k(w_i)$, we get that:
$\forall j\colon 1 \leq j \leq k+1\colon M_e(T^j(w_i)) \not\sqsubseteq \ell \wedge M'_e(T^j(w_i)) \not\sqsubseteq \ell$.
Thus, $\tau|_\ell =_{obs} \epsilon$ and $\tau'|_\ell =_{obs} \epsilon$.

We now prove $M_t|_\ell = M'_t|_\ell$. Because $M_t(cc) = M'_t(cc)$ and because bc is not modified, it suffices to prove $M_t|_\ell = M'_t|_\ell$ for the label chain of w_i. We prove for j with $1 \leq j \leq k+1$ that if $M_t(T^j(w_i)) \sqsubseteq \ell$, then $M'_t(T^j(w_i)) \sqsubseteq \ell$ and $M_t(T^{j-1}(w_i)) = M'_t(T^{j-1}(w_i))$. (III) and (IV) give $\forall j\colon 1 \leq j \leq i\colon T^j(w_i) \not\sqsubseteq \ell$. It then suffices to prove that the following holds:
$\forall j\colon i \leq j \leq k+1\colon M_t(T^j(w_i)) = M'_t(T^j(w_i))$.

For $j = i$, we have $M_t(T^j(w_i)) = M(T(a)) = M'_t(T^j(w_i))$. For j with $i < j < k+1$, we have $M_t(T^j(w_i)) = \bot = M'_t(T^j(w_i))$. Because $T^{k+1}(w_i) = T^k(w_i)$, we also have $M_t(T^{k+1}(w_i)) = \bot = M'_t(T^{k+1}(w_i))$. So, $M_t|_\ell = M'_t|_\ell$ holds.

So, BNI+(k-*Eopt*,$\mathcal{L}$,C) holds. Because BNI+(k-*Eopt*,$\mathcal{L}$,C) implies that k-BNI(k-*Eopt*,$\mathcal{L}$,C) holds, we get that k-BNI(k-*Eopt*,$\mathcal{L}$,C) holds, too. □

E Permissiveness of k-*Enf* versus Chain Length

Theorem 3. *k-Enf $<^{k,\mathcal{L}}_{\rho_k}$ $(k+1)$-Enf, for $k \geq 2$ and any lattice $\mathcal{L}$ with at least one non-bottom element.*

Proof. We first prove k-*Enf* $\leq^{k,\mathcal{L}}_{\rho_k}$ $(k{+}1)$-*Enf*. Consider $\tau = \mathit{trace}_{k\text{-}Enf}(C, M)$ with $M \models \mathcal{H}_0(k\text{-}Enf, \mathcal{L}, C)$. Consider M' such that $M' \models \mathcal{H}_0((k{+}1)\text{-}Enf, \mathcal{L}, C)$, $M|^k = M'|^k$, and $\tau' = \mathit{trace}_{(k+1)\text{-}Enf}(C, M')$. Using induction on the number of steps in τ and Lemma 13, we get that $\tau|^k_\ell \preceq \tau'|^k_\ell$, for all $\ell \in \mathcal{L}$. So, k-*Enf* $\leq^{k,\mathcal{L}}_{\rho_k}$ $(k+1)$-*Enf*.

To prove k-*Enf* $<^{k,\mathcal{L}}_{\rho_k}$ $(k+1)$-*Enf*, it suffices to also show that $(k+1)$-*Enf* $\not\leq^{k,\mathcal{L}}_{\rho_k}$ k-*Enf*. Because $\mathcal{L}$ contains at least one non-bottom element ℓ, with $\bot \sqsubset \ell$, there exists M_1 such that $M_1 \models \mathcal{H}_0((k+1)\text{-}Enf, \mathcal{L}, C)$,
$\tau = \mathit{trace}_{(k+1)\text{-}Enf}(w := w_1, M_1) = \langle w := w_1, M_1\rangle \to \langle \mathbf{stop}, M_2\rangle$,
and $M_1(T^{k+1}(w_1)) \sqsubset M_1(T^k(w_1))$.
There exists M'_1 such that $M'_1 \models \mathcal{H}_0(k\text{-}Enf, \mathcal{L}, C)$, $M_1|^k = M'_1|^k$, and
$\tau' = \mathit{trace}_{k\text{-}Enf}(w := w_1, M'_1) = \langle w := w_1, M'_1\rangle \to \langle \mathbf{stop}, M'_2\rangle$.
From $M_1|^k = M'_1|^k$, we have $M'_1(T^k(w_1)) = M_1(T^k(w_1))$. By definition of k-*Enf* (i.e., $\forall x\colon T^{k+1}(x) = T^k(x)$), we then have

$M_1'(T^{k+1}(w_1)) = M_1'(T^k(w_1)) = M_1(T^k(w_1)) \sqsupset M_1(T^{k+1}(w_1))$,
and thus, we get $M_2'(T^{k+1}(w)) \sqsupset M_2(T^{k+1}(w))$. So, τ generates observation involving $T^k(w)$ to label $M_2(T^{k+1}(w))$, but τ' does not generate observation involving $T^k(w)$ to label $M_2(T^{k+1}(w))$. So, $(k+1)$-*Enf* $\not\leq^{k,\mathcal{L}}_{\rho_k}$ k-*Enf*. Thus, k-*Enf* $<^{k,\mathcal{L}}_{\rho_k}$ $(k+1)$-*Enf*. □

Lemma 13. *Consider step $\tau = \langle C, M_1\rangle \to \langle C_2, M_2\rangle$ generated by k-Enf for $k \geq 2$. Consider step $\tau' = \langle C, M_1'\rangle \to \langle C_2', M_2'\rangle$ generated by $(k+1)$-Enf. If $M_1 =_k M_1'$, then $C_2 = C_2'$, $M_2 =_k M_2'$, and $\tau|^k_\ell \preceq \tau'|^k_\ell$, for all ℓ.*

Proof. By structural induction on C. □

Theorem 4. *k-Enf $\cong^{k,\mathcal{L}}_c$ $(k+1)$-Enf for any lattice $\mathcal{L}$ and $k \geq 2$*

Proof. We prove k-*Enf* $\leq^{k,\mathcal{L}}_c$ $(k+1)$-*Enf* and $(k+1)$-*Enf* $\leq^{k,\mathcal{L}}_c$ k-*Enf*. Consider conventionally initialized memory M with $M \models \mathcal{H}_0(k\text{-}Enf, \mathcal{L}, C)$ and $\tau = trace_{k\text{-}Enf}(C, M)$. Consider M' such that $M' \models \mathcal{H}_0((k{+}1)\text{-}Enf, \mathcal{L}, C)$, $\rho_1(M, M')$, and $\tau' = trace_{(k+1)\text{-}Enf}(C, M')$. So, M' is conventionally initialized, too. Thus, we have *2stut*(M) and *2stut*(M'). Also, because M and M' are conventionally initialized and $\rho_1(M, M')$ holds, we get that $M =_k M'$ holds. Using induction on the number of steps in τ and Lemma 14, we get that $\tau|^k_\ell = \tau'|^k_\ell$, for all $\ell \in \mathcal{L}$. So, k-*Enf* $\leq^{k,\mathcal{L}}_c$ $(k+1)$-*Enf* and $(k+1)$-*Enf* $\leq^{k,\mathcal{L}}_c$ k-*Enf*. Thus, k-*Enf* $\cong^{k,\mathcal{L}}_c$ $(k+1)$-*Enf*. □

Lemma 14. *Consider step $\tau = \langle C, M_1\rangle \to \langle C_2, M_2\rangle$ generated by k-Enf for $k \geq 2$. Consider step $\tau' = \langle C, M_1'\rangle \to \langle C_2', M_2'\rangle$ generated by $(k+1)$-Enf. If $M_1 =_k M_1'$, 2stut(M_1), and 2stut(M_1'), then $C_2 = C_2'$, $M_2 =_k M_2'$, 2stut(M_2), 2stut(M_2'), and $\tau|^k_\ell = \tau'|^k_\ell$, for all ℓ.*

Proof. By structural induction on C. □

Theorem 5. *k-Enf $\cong^{0,\mathcal{L}}_{\rho_k}$ $(k+1)$-Enf for any lattice $\mathcal{L}$ and $k \geq 2$.*

Proof. Lemma 15 gives *2-Enf* $\cong^{0,\mathcal{L}}_{\rho_2}$ k-*Enf* for any lattice $\mathcal{L}$ and $k \geq 2$. Because $\rho_k \Rightarrow \rho_2$, we then get *2-Enf* $\cong^{0,\mathcal{L}}_{\rho_k}$ k-*Enf* for $k \geq 2$. By transitivity, we then have k-*Enf* $\cong^{0,\mathcal{L}}_{\rho_k}$ $(k+1)$-*Enf*. □

Lemma 15. *2-Enf $\cong^{0,\mathcal{L}}_{\rho_2}$ k-Enf for any lattice $\mathcal{L}$ and $k > 2$.*

Proof. We prove k-*Enf* $\leq^{0,\mathcal{L}}_{\rho_2}$ 2-*Enf* and 2-*Enf* $\leq^{0,\mathcal{L}}_{\rho_2}$ k-*Enf*. Consider memory M with $M \models \mathcal{H}_0(2\text{-}Enf, \mathcal{L}, C)$ and $\tau = trace_{2\text{-}Enf}(C, M)$. Consider memory

M' such that $M' \models \mathcal{H}_0(k\text{-}Enf, \mathcal{L}, C)$, $\rho_2(M, M')$, and $\tau' = trace_{k\text{-}Enf}(C, M')$. Using induction on the number of steps in τ and Lemma 16, we get that $\tau|^0_\ell = \tau'|^0_\ell$, for all $\ell \in \mathcal{L}$. So, $k\text{-}Enf \preceq^{0,\mathcal{L}}_{\rho_2} 2\text{-}Enf$ and $2\text{-}Enf \preceq^{0,\mathcal{L}}_{\rho_2} k\text{-}Enf$. Thus, $2\text{-}Enf \cong^{0,\mathcal{L}}_{\rho_2} k\text{-}Enf$. □

Lemma 16. *Consider step $\tau = \langle C, M_1\rangle \to \langle C_2, M_2\rangle$ generated by k-Enf for $k \geq 2$. Consider step $\tau' = \langle C, M_1'\rangle \to \langle C_2', M_2'\rangle$ generated by 2-Enf. If $M_1 =_2 M_1'$, then $C_2 = C_2'$, $M_2 =_2 M_2'$, and $\tau|^0_\ell = \tau'|^0_\ell$, for all ℓ.*

Proof. By structural induction on C. □

F Other Enforcers

F.1 Strong Threat Model

Theorem 6. *For a lattice $\mathcal{L}$, for an enforcer E that satisfies $(k-1)$-BNI($\mathcal{L}$), with $k \geq 2$, and produces some k-precise k-varying label chains with elements in $\mathcal{L}$, and for an enforcer E' that produces $(k-1)$-dependent label chains,*

$$\text{if } E \preceq^{k-1,\mathcal{L}}_{c} E', \text{ then } E' \text{ does not satisfy } (k-1)\text{-BNI}(\mathcal{L}).$$

Enforcer E and lattice $\mathcal{L}$ exist.

Proof. First we prove that E and $\mathcal{L}$ exist. Lemma 17 gives that k-*Eopt* is an enforcer, satisfies $(k-1)$-BNI($\mathcal{L}_k$), and produces some k-precise k-varying label chains with elements in $\mathcal{L}_k$, which is defined in (31). So, $\mathcal{L}$ exists and it can be $\mathcal{L}_k$, and E exists and it can be k-*Eopt*.

Assume a lattice $\mathcal{L}$ and an enforcer E that satisfies $(k-1)$-BNI($\mathcal{L}$) and produces some k-precise k-varying label chains with elements in $\mathcal{L}$:

$$\Omega = \langle \ell_1, \ell_2, \ldots, \ell_k\rangle \text{ and } \Omega' = \langle \ell_1, \ell_2, \ldots, \ell_k'\rangle$$

with $\ell_k \neq \ell_k'$. Assume an enforcer E' that produces $(k-1)$-dependent label chains and $E \preceq^{k-1,\mathcal{L}}_{c} E'$.

We prove that E' does not satisfy $(k-1)$-BNI($\mathcal{L}$). Assume for contradiction that E' satisfies $(k-1)$-BNI($\mathcal{L}$). We have that there are j, C, and $M \models \mathcal{H}_0(E, \mathcal{L}, C)$ such that Ω is k-precise at the jth state of $\tau = trace_E(C, M)$. There exists a memory M_1 such that M_1 is conventionally initialized, $M_1 \models \mathcal{H}_0(E', \mathcal{L}, C)$ holds, and $\rho_1(M, M_1)$. Let $\tau_1 = trace_{E'}(C, M_1)$.

By definition of k-precise and because $E \leq_c^{k-1,\mathcal{L}} E'$ and E' satisfies $(k-1)$-BNI($\mathcal{L}$), we then get that τ_1 produces Ω at the jth state. So, by definition, $1 \leq j \leq |\tau_1|$ and there exists w such that:

$$\tau_1[j-1] = \langle w := e; C_r, M_w\rangle,\ \tau_1[j] = \langle C_r, M_r\rangle,$$
$$\forall i\colon 1 \leq i \leq k\colon M_r(T^i(w)) = \ell_i.$$

Working similarly for Ω', we get:

$$\tau_2[s-1] = \langle w' := e'; C'_r, M'_w\rangle,\ \tau_2[s] = \langle C'_r, M'_r\rangle,$$
$$\forall i\colon 1 \leq i < k\colon M'_r(T^i(w')) = \ell_i,\ M'_r(T^k(w')) = \ell'_k$$

for $\tau_2 = \mathit{trace}_{E'}(C', M_2)$, conventionally initialized memory M_2 with $M_2 \models \mathcal{H}_0(E', \mathcal{L}, C')$, and $1 \leq s \leq |\tau_2|$. Because E' uses $(k-1)$-dependent label chains, there exists a function f such that:

$$M_r(T^k(w)) = f(M_r(T(w)), \ldots, M_r(T^{k-1}(w)))$$
$$M'_r(T^k(w')) = f(M'_r(T(w')), \ldots, M'_r(T^{k-1}(w'))).$$

Because $\forall i\colon 1 \leq i < k\colon M_r(T^i(w)) = M'_r(T^i(w')) = \ell_i$, we then have that $M_r(T^k(w)) = M'_r(T^k(w'))$ holds. But $M_r(T^k(w)) = \ell_k$, $M'_r(T^k(w')) = \ell'_k$, and $\ell_k \neq \ell'_k$ give $M_r(T^k(w)) \neq M'_r(T^k(w'))$, which is a contradiction. $\square$

Lemma 17. *For $k \geq 2$, k-Eopt is an enforcer, satisfies $(k-1)$-BNI($\mathcal{L}_k$), and produces k-precise k-varying label chains with elements in $\mathcal{L}_k$, which is defined in (31).*

Proof. Lemma 12 gives that *k-Eopt* is an enforcer on R for k-BNI. Thus, *k-Eopt* satisfies $(k-1)$-BNI($\mathcal{L}_k$).

Lemma 18 gives the possible label chains that *k-Eopt* produces for each z_j in pgm_k, which is defined below. Lemma 19 gives that these label chains are k-precise. The last label chain in (Z_{k-1}) is $\langle \ell_{k-1}, \ell_{k-1}, \ldots, \ell_{k-1}, \perp\rangle$ and has length k. The penultimate label chain in (Z_k) is $\langle \ell_{k-1}, \ell_{k-1}, \ldots, \ell_{k-1}, \ell_k\rangle$ and has length k. The above two label chains take elements from $\mathcal{L}_k$ and they are k-varying. $\square$

Definition of pgm_k for $k \geq 2$

Let pgm_k be the following program:

> **if** $a_1 > 0$ **then** $w_1 := 0$ **else** $w_1 := 1$ **end**;
> $z_1 := w_1$;
> **if** $a_2 > 0$ **then** $w_2 := z_1$ **else** $w_2 := 2$ **end**;
> $z_2 := w_2$;
> $\ldots$
> **if** $a_{k-1} > 0$ **then** $w_{k-1} := z_{k-2}$ **else** $w_{k-1} := k-1$ **end**;
> $z_{k-1} := w_{k-1}$;
> **if** $a_k > 0$ **then** $w_k := z_{k-1}$ **else** $w_k := k$ **end**;
> $z_k := w_k$;

where all w_k and z_k are flexible variables. Assume lattice $\mathcal{L}_k$ of labels such that

$$\ell_0 \sqsupset \ell_1 \sqsupset \ell_2 \sqsupset \ldots \sqsupset \ell_k \sqsupset \bot \tag{31}$$

Assume $\mathcal{L}_k$ consists only of $\bot, \ell_j$, for $0 \leq j \leq k$. Assume pgm_k is executed with conventionally initialized memory M under k-*Eopt*, where $M(T(a_j)) = \ell_j$, for $0 \leq j \leq k$ and $k \geq 2$.

($\mathbf{Z_1}$) After the execution of $z_1 := w_1$, this is the possible chain for z_1:

$$\begin{array}{lllll} & T(z_1) & T^2(z_1) & \ldots & T^k(z_1) & \\ \langle & \ell_1 & \bot & \ldots & \bot & \rangle \end{array}$$

($\mathbf{Z_2}$) After the execution of $z_2 := w_2$, these are the possible 2 chains for z_2:

$$\begin{array}{llllll||l} & T(z_2) & T^2(z_2) & T^3(z_2) & \ldots & & \\ \langle & \ell_1 & \ell_2 & \bot & \ldots & \rangle & a_2 > 0 \\ \langle & \ell_2 & \ell_2 & \bot & \ldots & \rangle & a_2 \not> 0 \end{array}$$

($\mathbf{Z_3}$) After the execution of $z_3 := w_3$, these are the possible 3 chains for z_3:

$$\begin{array}{lllllll||l} & T(z_3) & T^2(z_3) & T^3(z_3) & T^4(z_3) & \ldots & & \\ \langle & \ell_1 & \ell_2 & \ell_3 & \bot & \ldots & \rangle & a_2 > 0 \wedge a_3 > 0 \\ \langle & \ell_2 & \ell_2 & \ell_3 & \bot & \ldots & \rangle & a_2 \not> 0 \wedge a_3 > 0 \\ \langle & \ell_3 & \ell_3 & \ell_3 & \bot & \ldots & \rangle & a_3 \not> 0 \end{array}$$

...

($\mathbf{Z_j}$) After the execution of $z_j := w_j$, these are the possible j chains for z_j:

$$
\begin{array}{llllll|l}
T & T^2 & T^3 & \ldots T^{j-1} & T^j & T^{j+1}\ldots & \\
\langle \ell_1 & \ell_2 & \ell_3 & \ldots \ell_{j-1} & \ell_j & \bot \ldots\rangle & a_2, a_3, \ldots, a_j > 0 \\
\langle \ell_2 & \ell_2 & \ell_3 & \ldots \ell_{j-1} & \ell_j & \bot \ldots\rangle & a_2 \not> 0 \wedge a_3, \ldots, a_j > 0 \\
\langle \ell_3 & \ell_3 & \ell_3 & \ldots \ell_{j-1} & \ell_j & \bot \ldots\rangle & a_3 \not> 0 \wedge a_4, \ldots, a_j > 0 \\
\ldots & & & & & & \\
\langle \ell_{j-1} & \ell_{j-1} & \ell_{j-1} & \ldots \ell_{j-1} & \ell_j & \bot \ldots\rangle & a_{j-1} \not> 0 \wedge a_j > 0 \\
\langle \ell_j & \ell_j & \ell_j & \ldots \ell_j & \ell_j & \bot \ldots\rangle & a_j \not> 0
\end{array}
$$

($\mathbf{Z_k}$) After the execution of $z_k := w_k$, these are the possible k chains for z_i:

$$
\begin{array}{llllll|l}
T & T^2 & T^3 & \ldots & T^{k-1} & T^k & \\
\langle \ell_1 & \ell_2 & \ell_3 & \ldots & \ell_{k-1} & \ell_k\rangle & a_2 > 0 \wedge a_3 > 0 \wedge \ldots \wedge a_j > 0 \\
\langle \ell_2 & \ell_2 & \ell_3 & \ldots & \ell_{k-1} & \ell_k\rangle & a_2 \not> 0 \wedge a_3 > 0 \wedge \ldots \wedge a_k > 0 \\
\langle \ell_3 & \ell_3 & \ell_3 & \ldots & \ell_{k-1} & \ell_k\rangle & a_3 \not> 0 \wedge \ldots \wedge a_k > 0 \\
\ldots & & & & & & \\
\langle \ell_{k-1} & \ell_{k-1} & \ell_{k-1} & \ldots & \ell_{k-1} & \ell_k\rangle & a_{k-1} \not> 0 \wedge a_k > 0 \\
\langle \ell_k & \ell_k & \ell_k & \ldots & \ell_k & \ell_k\rangle & a_k \not> 0
\end{array}
$$

Lemma 18. *The label chains presented after pgm_k are the only possible chains that k-Eopt produces for variables z_j, where $k \geq 2$ and $1 \leq j \leq k$.*

Proof. Induction on j.

Base case: $j = 1$.
When execution reaches C_1:

$$\textbf{if } a_1 > 0 \textbf{ then } w_1 := 0 \textbf{ else } w_1 := 1 \textbf{ end}$$

with a memory M', then *isSimple*$(C_1, M', 1)$ is always satisfied. Using IfS, Exit_IfS, and AsgnF rules, we get that z_1 is always associated with: $\langle \ell_1, \bot, \ldots, \bot\rangle$.

Induction case:
IH: chains presented after *pgm*$_k$ for z_{j-1}, with $j > 1$, are the only possible chains that k-Eopt produces for z_{j-1}.

We prove that chains presented after *pgm*$_k$ are the only possible chains that k-Eopt produces for z_j. When execution reaches C_j:

$$\textbf{if } a_j > 0 \textbf{ then } w_j := z_{j-1} \textbf{ else } w_j := j \textbf{ end}$$

with some memory M', then using IH on z_{j-1} we get that *isSimple*(C_j, M', j) is always satisfied. So, rules IfS and Exit_IfS are used while executing C_j.

1. $a_j > 0$
   Assignment $w_j := z_{j-1}$ is executed. From IH, IFS and ASGNF, we have that w_j is associated after $w_j := z_{j-1}$ with one of the following $j-1$ label chains:

$$\begin{array}{llllllll|l}
T & T^2 & T^3 & \ldots T^{j-1} & T^j & T^{j+1} \ldots & & & \\
\langle \ell_1 & \ell_2 & \ell_3 & \ldots \ell_{j-1} & \ell_j & \ell_j \ldots \rangle & & & a_2, a_3, \ldots, \\
 & & & & & & & & a_{j-1} > 0 \\
\langle \ell_2 & \ell_2 & \ell_3 & \ldots \ell_{j-1} & \ell_j & \ell_j \ldots \rangle & & & a_2 \not> 0 \wedge a_3, \ldots, \\
 & & & & & & & & a_{j-1} > 0 \\
\langle \ell_3 & \ell_3 & \ell_3 & \ldots \ell_{j-1} & \ell_j & \ell_j \ldots \rangle & & & a_3 \not> 0 \wedge a_4, \ldots, \\
 & & & & & & & & a_{j-1} > 0 \\
\ldots & & & & & & & & \\
\langle \ell_{j-1} & \ell_{j-1} & \ell_{j-1} & \ldots \ell_{j-1} & \ell_j & \ell_j \ldots \rangle & & & a_{j-1} \not> 0
\end{array}$$

   From EXIT_IFS, we have that w_j is associated at the end of C_j with one of the following $j-1$ label chains:

$$\begin{array}{llllllll|l}
T & T^2 & T^3 & \ldots & T^{j-1} & T^j & T^{j+1} \ldots & & \\
\langle \ell_1 & \ell_2 & \ell_3 & \ldots & \ell_{j-1} & \ell_j & \perp \ldots \rangle & & a_2, a_3, \ldots, \\
 & & & & & & & & a_{j-1} > 0 \\
\langle \ell_2 & \ell_2 & \ell_3 & \ldots & \ell_{j-1} & \ell_j & \perp \ldots \rangle & & a_2 \not> 0 \wedge \\
 & & & & & & & & a_3, ..., a_{j-1} > 0 \\
\langle \ell_3 & \ell_3 & \ell_3 & \ldots & \ell_{j-1} & \ell_j & \perp \ldots \rangle & & a_3 \not> 0 \wedge \\
 & & & & & & & & a_4, ..., a_{j-1} > 0 \\
\ldots & & & & & & & & \\
\langle \ell_{j-1} & \ell_{j-1} & \ell_{j-1} & \ldots & \ell_{j-1} & \ell_j & \perp \ldots \rangle & & a_{j-1} \not> 0
\end{array}$$

2. $a_j \not> 0$
   Assignment $w_j := j$ is executed. From IFS and ASGNF, we have that w_j is associated after $w_j := j$ with the following label chain:

$$\begin{array}{cccccccccc|}
 & T & T^2 & T^3 & \ldots & T^{j-1} & T^j & T^{j+1} & \ldots & \\
\langle & \ell_j & \ell_j & \ell_j & \ldots & \ell_j & \ell_j & \ell_j & \ldots & \rangle
\end{array}$$

   From EXIT_IFS, we have that w_j is associated at the end of C_j with the following label chain:

$$\begin{array}{cccccccccc|}
 & T & T^2 & T^3 & \ldots & T^{j-1} & T^j & T^{j+1} & \ldots & \\
\langle & \ell_j & \ell_j & \ell_j & \ldots & \ell_j & \ell_j & \perp & \ldots & \rangle
\end{array}$$

So, using $\textsc{AsgnF}$ and the j above possible label chains produced for w_j at the end of C_j, we get that label chains presented after pgm_k for z_j are the only possible label chains that k-Eopt produces for z_j. $\square$

Lemma 19. *The label chains produced by k-Eopt for each z_j in pgm_k are k-precise, where $k \geq 2$ and $1 \leq j \leq k$.*

Proof. Consider j such that $1 \leq j \leq k$ and $k \geq 2$. We prove that the label chains for z_j produced by *k-Eopt* are k-precise. We use induction on the number n of elements in these label chains, where $1 \leq n \leq k$.

Base case: $n = 1$

We prove that the label chains produced by *k-Eopt* for z_j are 1-precise. Consider $\tau = trace_{k\text{-}Eopt}(pgm_k, M)$, where M is conventionally initialized and $M \models \mathcal{H}_0(k\text{-}Eopt, \mathcal{L}_k, pgm_k)$. Then $\tau[s-1] = \langle z_j := w_j; C, M_0\rangle$ and $\tau[s] = \langle C, M_1\rangle$ for some $1 < s \leq |\tau|$. Trace τ produces label chain $\Omega = \langle M_1(T(z_j)), M_1(T^2(z_j)), \ldots, M_1(T^k(z_j))\rangle$ at the sth state.

We prove that Ω is 1-precise. Ω may be one of the j possible label chains in $(\mathbf{Z_j})$. So, the only possible labels for $M_1(T(z_j))$ are $\ell_1, \ell_2, \ldots, \ell_j$. Let $M_1(T(z_j)) = \ell_i$ for $1 \leq i \leq j$. Only the ith label chain in $(\mathbf{Z_j})$ has $T(z_j) = \ell_i$. So, it should be the case that

$$\begin{aligned} M(a_i) \not> 0, M(a_{i+1}) > 0, \ldots M(a_j) > 0 \quad & \text{if } i \neq j \\ M(a_j) \not> 0 \quad & \text{if } i = j \end{aligned} \tag{32}$$

Consider an enforcer D that satisfies 0-BNI($\mathcal{L}_k$) and $k\text{-}Eopt \leq_c^{0,\mathcal{L}_k} D$. Consider memory M' with $M' \models \mathcal{H}_0(D, \mathcal{L}_k, pgm_k)$, $\rho_1(M, M')$ and also $\tau' = trace_D(pgm_k, M')$. It should be the case that $\tau'[s] = \langle C, M_1'\rangle$, because otherwise $\forall \ell \in \mathcal{L}_k$: $\tau|_\ell^0 \trianglelefteq \tau'|_\ell^0$ (and $k\text{-}Eopt \leq_c^{0,\mathcal{L}_k} D$) would not hold.

We prove $M_1'(T(z_j)) = M_1(T(z_j)) = \ell_i$. Assume for contradiction that $M_1'(T(z_j)) \neq M_1(T(z_j))$. Either $M_1'(T(z_j)) \not\sqsubseteq M_1(T(z_j))$ or $M_1'(T(z_j)) \sqsubset M_1(T(z_j))$ holds. From $M_1'(T(z_j)) \not\sqsubseteq M_1(T(z_j))$ and $M_1'(T(z_j)) \neq M_1(T(z_j))$, we get $\tau|_{\ell_i}^0 \not\trianglelefteq \tau'|_{\ell_i}^0$, which implies that $k\text{-}Eopt \not\leq_c^{0,\mathcal{L}_k} D$, which is a contradiction. Assume $M_1'(T(z_j)) \sqsubset M_1(T(z_j))$. From $M_1'(T(z_j)) \sqsubset M_1(T(z_j))$ and $M_1(T(z_j)) = \ell_i$, we then have $M_1'(T(z_j)) \sqsubset \ell_i$. Let $\ell = M_1'(T(z_j))$. There exists M''' such that $M''' \models \mathcal{H}_0(k\text{-}Eopt, \mathcal{L}_k, pgm_k)$ and

$$M \text{ and } M''' \text{ agree on everything except for } a_i > 0 \tag{33}$$

So M''' is conventionally initialized.

Let $\tau''' = trace_{k\text{-}Eopt}(pgm_k, M''')$. We have $\tau'''[s-1] = \langle z_j := w_j; C, M_0''' \rangle$ and $\tau'''[s] = \langle C, M_1''' \rangle$. There exists M'' such that $M'' \models \mathcal{H}_0(D, \mathcal{L}_k, pgm_k)$ and $\rho_1(M''', M'')$ hold.

Let $\tau'' = trace_D(pgm_k, M'')$. We should have $\tau''[s-1] = \langle z_j := w_j; C, M_0'' \rangle$ and $\tau''[s] = \langle C, M_1'' \rangle$, because otherwise $\forall \ell \in \mathcal{L}_k$: $\tau'''|_\ell^0 \trianglelefteq \tau''|_\ell^0$ (and k-$Eopt \leq_c^{0,\mathcal{L}_k} D$) would not hold.

Because M is conventionally initialized, and because we have $\rho_1(M, M')$, $\rho_1(M''', M'')$, and (33), we get that M' and M'' agree on everything except for $a_i > 0$. In particular, from $\rho_1(M, M')$ and (32) we get $M'(T(a_i)) = M(T(a_i)) = \ell_i$ and

$$\begin{aligned} M'(a_i) \not> 0, M'(a_{i+1}) > 0, \ldots M'(a_j) > 0 \text{ if } i \neq j \\ M'(a_j) \not> 0 \text{ if } i = j \end{aligned} \tag{34}$$

From (32), (33), and $\rho_1(M''', M'')$, we get

$$\begin{aligned} M''(a_i) > 0, M''(a_{i+1}) > 0, \ldots M''(a_j) > 0 \text{ if } i \neq j \\ M''(a_j) > 0 \text{ if } i = j \end{aligned} \tag{35}$$

Because M' and M'' agree on everything except for a_i and because we have $M'(T(a_i)) = \ell_i \not\sqsubseteq \ell$, we get $M'|_\ell = M''|_\ell$. So, based on pgm_k, we get $M_1''(z_j) < i$ and $M_1'(z_j) = i$ for $1 \leq i \leq j$. Thus, $M_1'(z_j) \neq M_1''(z_j)$.

Because $M_1'(T(z_j)) = \ell$, observation $\langle z_j, M_1'(z_j) \rangle$ appears in $\tau'|_\ell^0$. If $M_1''(z_j) \not\sqsubseteq \ell$, then observation $\langle z_j, M_1''(z_j) \rangle$ does not appear in $\tau''|_\ell^0$, and thus D does not satisfy 0-BNI($\mathcal{L}_k$). If $M_1''(z_j) \sqsubseteq \ell$, then observation $\langle z_j, M_1''(z_j) \rangle$ appears in $\tau''|_\ell^0$. But because $M_1'(z_j) \neq M_1''(z_j)$, D does not satisfy 0-BNI($\mathcal{L}_k$). So, in any case D does not satisfy 0-BNI($\mathcal{L}_k$), which is a contradiction. So, $M_1'(T(z_j)) = M_1(T(z_j))$. Thus, the label chain produced by k-$Eopt$ for z_j is 1-precise.

Induction case:
IH: The label chains for z_j are n-precise, for $1 \leq n < k$.
We prove that the label chains for z_j are $(n+1)$-precise.

If $n + 1 \geq j + 1$, then $T^{n+1}(z_j) = \perp$, and thus using IH we get that the label chains for z_j are $(n+1)$-precise.

Assume $1 \leq n < j$. Consider $\tau = trace_{k\text{-}Eopt}(pgm_k, M)$ for a conventionally initialized memory M with $M \models \mathcal{H}_0(k\text{-}Eopt, \mathcal{L}_k, pgm_k)$. Then $\tau[s-1] = \langle z_j := w_j; C, M_0 \rangle$ and $\tau[s] = \langle C, M_1 \rangle$ for some $1 < s \leq |\tau|$. Trace τ produces label chain $\Omega = \langle M_1(T(z_j)), M_1(T^2(z_j)), \ldots, M_1(T^k(z_j)) \rangle$ at the sth state.

We prove that Ω is $(n+1)$-precise. Ω may be one of the j possible label chains in ($\mathbf{Z_j}$). So, we get $M_1(T^{n+1}(z_j)) = \ell_m$, for $1 \leq m \leq j$. Consider an enforcer D that satisfies n-BNI($\mathcal{L}_k$) and $k\text{-}Eopt \leq_c^{n,\mathcal{L}_k} D$. Assume $\tau' = trace_D(pgm_k, M')$, for memory M' with $M' \models \mathcal{H}_0(D, \mathcal{L}_k, pgm_k)$ and $\rho_1(M, M')$. It should be the case that $\tau'[s-1] = \langle z_j := w_j; C, M'_0\rangle$ and $\tau'[s] = \langle C, M'_1\rangle$, because otherwise $k\text{-}Eopt \leq_c^{n,\mathcal{L}_k} D$ would not hold. From IH, n-BNI($\mathcal{L}_k, D$), and $k\text{-}Eopt \leq_c^{n,\mathcal{L}_k} D$ we get that

$$\forall t\colon 1 \leq t \leq n\colon M'_1(T^t(z_j)) = M_1(T^t(z_j)). \tag{36}$$

We prove $M'_1(T^{n+1}(z_j)) = M_1(T^{n+1}(z_j))$. Assume for contradiction that $M'_1(T^{n+1}(z_j)) \neq M_1(T^{n+1}(z_j))$. Either $M'_1(T^{n+1}(z_j)) \not\sqsubset M_1(T^{n+1}(z_j))$ or $M'_1(T^{n+1}(z_j)) \sqsubset M_1(T^{n+1}(z_j))$ holds. From $M'_1(T^{n+1}(z_j)) \not\sqsubset M_1(T^{n+1}(z_j))$ and $M'_1(T^{n+1}(z_j)) \neq M_1(T^{n+1}(z_j))$, we get $k\text{-}Eopt \not\leq_c^{n,\mathcal{L}_k} D$, which is a contradiction.

Assume $M'_1(T^{n+1}(z_j)) \sqsubset M_1(T^{n+1}(z_j))$. So $M'_1(T^{n+1}(z_j)) \sqsubset \ell_m$. Let $\ell = M'_1(T^{n+1}(z_j))$. There exists memory M_m such that $M_m \models \mathcal{H}_0(k\text{-}Eopt, \mathcal{L}_k, pgm_k)$ and $M_m = M[\neg(M(a_m) > 0)]$, which denotes that M_m and M agree on everything expect for $a_m > 0$: $(M_m(a_m) > 0) = \neg(M(a_m) > 0)$. So, M_m is conventionally initialized.

Let $\tau_m = trace_{k\text{-}Eopt}(pgm_k, M_m)$. So, $\tau_m[s-1] = \langle z_j := w_j; C, M_{m0}\rangle$ and $\tau_m[s] = \langle C, M_{m1}\rangle$. From Lemma 20, we get

$$M_{m1}(T^n(z_j)) \neq M_1(T^n(z_j)). \tag{37}$$

There exists M'' with $M'' \models \mathcal{H}_0(D, \mathcal{L}_k, pgm_k)$, $\rho_1(M_m, M'')$, and $\tau'' = trace_D(pgm_k, M'')$. We should have $\tau''[s-1] = \langle z_j := w_j; C, M''_0\rangle$ and $\tau''[s] = \langle C, M''_1\rangle$, because otherwise $\forall \ell \in \mathcal{L}_k\colon \tau_m|_\ell^n \trianglelefteq \tau''|_\ell^n$ (and $k\text{-}Eopt \leq_c^{n,\mathcal{L}_k} D$) would not hold. From $\rho_1(M_m, M'')$, $\rho_1(M, M')$, and $M_m = M[\neg(M(a_m) > 0)]$, and because M, M_m are conventionally initialized, we get that M' and M'' agree on everything except for a_m: $M'(a_m > 0) = \neg M''(a_m > 0)$. From $\rho_1(M, M')$ and $M(T(a_m)) = \ell_m$, we get $M'(T(a_m)) = \ell_m$ and then $M''(T(a_m)) = \ell_m$. From $\ell_m \not\sqsubseteq \ell$, we then get $M'|_\ell = M''|_\ell$. By IH, n-BNI($\mathcal{L}_k, D$), and $k\text{-}Eopt \leq_c^{n,\mathcal{L}_k} D$ we get $M''_1(T^n(z_j)) = M_{m1}(T^n(z_j))$. From (36) and (37) we then get $M'_1(T^n(z_j)) \neq M''_1(T^n(z_j))$.

Because $M'_1(T^{n+1}(z_j)) = \ell$, observation $\langle T^n(z_j), M'_1(T^n(z_j))\rangle$ appears in $\tau'|_\ell^n$. If $M''_1(T^{n+1}(z_j)) \not\sqsubseteq \ell$, then observation $\langle T^n(z_j), M''_1(T^n(z_j))\rangle$ does not appear in $\tau''|_\ell^n$, and thus n-BNI($\mathcal{L}_k, D$) does not hold. If $M''_1(T^{n+1}(z_j)) \sqsubseteq \ell$, then observation $\langle T^n(z_j), M''_1(T^n(z_j))\rangle$ appears in $\tau''|_\ell^n$. But because we have

$M_1'(T^n(z_j)) \neq M_1''(T^n(z_j))$, n-BNI$(\mathcal{L}_k, D)$ does not hold. So, in any case n-BNI$(\mathcal{L}_k, D)$ does not hold, which is a contradiction. So, $M_1'(T^{n+1}(z_j)) = M_1(T^{n+1}(z_j))$. Thus, the label chains produced by *k-Eopt* for z_j are $(n+1)$-precise. $\square$

Lemma 20. *Consider $\tau = trace_{k\text{-}Eopt}(pgm_k, M)$, where M is conventionally initialized, $\tau[s-1] = \langle z_j := w_j; C, M_0\rangle$ and $\tau[s] = \langle C, M_1\rangle$. Assume $M_1(T^{n+1}(z_j)) = \ell_m$, with $1 \leq n < j$, $1 \leq j \leq k$, and $1 \leq m \leq j$. Then for $\tau' = trace_{k\text{-}Eopt}(pgm_k, M')$ with $M' = M[\neg(M(a_m > 0))]$, $\tau'[s-1] = \langle z_j := w_j; C, M_0'\rangle$, and $\tau'[s] = \langle C, M_1'\rangle$,*
we get $M_1(T^n(z_j)) \neq M_1'(T^n(z_j))$.

Proof. *1.* $M_1(T(z_j)) = \ell_m$

From $(\mathbf{Z_j})$ and $M_1(T^{n+1}(z_j)) = \ell_m$, we then get that $2 \leq n+1 \leq m$. So, $1 \leq n \leq m-1$. Also, $M_1(T^n(z_j)) = \ell_m$. Such a label chain $M_1(\Omega_{z_j})$ for z_j is generated when $M(a_m) \not> 0 \wedge M(a_{m+1}) > 0 \wedge \ldots \wedge M(a_j) > 0$. So, for M' we should have $M'(a_m) > 0 \wedge M'(a_{m+1}) > 0 \wedge \ldots \wedge M'(a_j) > 0$, because $M' = M[\neg(M(a_m > 0))]$. Thus, $M_1'(\Omega_{z_j})$ should be one of the label chains that appear above $M_1(\Omega_{z_j})$ in $(\mathbf{Z_j})$. From $1 \leq n \leq m-1$, we then have $M_1'(T^n(z_j)) \neq \ell_m$. So, $M_1(T^n(z_j)) \neq M_1'(T^n(z_j))$.

2. $M_1(T(z_j)) \neq \ell_m$

From $(\mathbf{Z_j})$ and $M_1(T^{n+1}(z_j)) = \ell_m$, we then have $M_1(T^n(z_j)) = \ell_{m-1}$ and $n = m-1$ (so $m \neq 1$). Also, M should have $M'(a_m) > 0 \wedge M'(a_{M+1}) > 0 \wedge \ldots \wedge M'(a_j) > 0$. So, M' should have $M'(a_m) \not> 0 \wedge M'(a_{M+1}) > 0 \wedge \ldots \wedge M'(a_j) > 0$, because $M' = M[\neg(M(a_m > 0))]$. Thus, $M_1'(\Omega_{z_j})$ is the mth label chain in $(\mathbf{Z_j})$. So, $M_1'(T^n(z_j)) = M_1'(T^{m-1}(z_j)) = \ell_m$. Thus, $M_1(T^n(z_j)) \neq M_1'(T^n(z_j))$.

$\square$

F.2 Weakened Threat Model

Theorem 7. *For an enforcer E and lattice $\mathcal{L}_3$, if $G^E_{a:=e}$ is 1-dependent and 2-Enf $\leq_c^{0,\mathcal{L}_3} E$, then E does not satisfy 0-BNI$(\mathcal{L}_3)$.*

Proof. We have $\mathcal{L} = \langle\{\mathsf{L}, \mathsf{M}, \mathsf{H}\}, \sqsubseteq\rangle$ where $\mathsf{L} \sqsubset \mathsf{M} \sqsubset \mathsf{H}$ and $\perp = \mathsf{L}$.

Consider program *pgm*:

$$
\begin{array}{l}
w_a := m_a; \\
\textbf{if } m_b > 0 \textbf{ then } w_b := m_b \textbf{ else } w_b := h \textbf{ end}; \\
\textbf{if } l > 1 \textbf{ then } w_c := w_a \textbf{ else } w_c := w_b \textbf{ end}; \\
w := w_c; \\
m := w; \\
l := 1
\end{array}
$$

where l, m_a, m_b, m, h are anchor variables with $T(l) = \mathsf{L}$, $T(m_a) = T(m_b) = T(m) = \mathsf{M}$, and $T(h) = \mathsf{H}$, and w, w_a, w_b, w_c are flexible variables.

2-*Enf* produces the following labels when executes *pgm* with a conventionally initialized memory:

	$m_b > 0$	$m_b \not> 0$
$l > 1$	$w_a : \langle \mathsf{M}, \mathsf{L} \rangle$ $w_b : \langle \mathsf{M}, \mathsf{M} \rangle$ $w : \langle \mathsf{M}, \mathsf{L} \rangle$	$w_a : \langle \mathsf{M}, \mathsf{L} \rangle$ $w_b : \langle \mathsf{H}, \mathsf{M} \rangle$ $w : \langle \mathsf{M}, \mathsf{L} \rangle$
$l \not> 1$	$w_a : \langle \mathsf{M}, \mathsf{L} \rangle$ $w_b : \langle \mathsf{M}, \mathsf{M} \rangle$ $w : \langle \mathsf{M}, \mathsf{M} \rangle$	$w_a : \langle \mathsf{M}, \mathsf{L} \rangle$ $w_b : \langle \mathsf{H}, \mathsf{M} \rangle$ $w : \langle \mathsf{H}, \mathsf{M} \rangle$

We first prove that, for all executions, $T(w)$ produced by 2-*Enf* is 1-precise.
Consider $\tau = \mathit{trace}_{2\text{-}Enf}(\mathit{pgm}, M)$, where M is conventionally initialized and $M \models \mathcal{H}_0(2\text{-}Enf, \mathcal{L}, \mathit{pgm})$. There exists s such that $\tau[s-1] = \langle w := w_c; C_r, M_w \rangle$ and $\tau[s] = \langle C_r, M_r \rangle$.

We prove that $M_r(T(w))$ is 1-precise. Consider E' an enforcer that satisfies 0-BNI($\mathcal{L}$) and 2-$E \leq_c^{0,\mathcal{L}} E'$. Assume $\tau' = \mathit{trace}_{E'}(\mathit{pgm}, M')$ where $M' \models \mathcal{H}_0(E', \mathcal{L}, \mathit{pgm})$ and

$$\rho_1(M, M'). \tag{38}$$

From 2-*Enf* $\leq_c^{0,\mathcal{L}} E'$, we get $\forall \ell \in \mathcal{L}$: $\tau|_\ell^0 \trianglelefteq \tau'|_\ell^0$. So, it should be the case that $\tau'[s-1] = \langle w := w_c; C_r, M'_w \rangle$ and $\tau'[s] = \langle C_r, M'_r \rangle$. We prove $M'_r(T(w)) = M_r(T(w))$.

1. $M(l) \not> 1$ and $M(m_b) > 0$ (a1)
We have $M_r(T(w)) = \mathsf{M}$. We prove $M'_r(T(w)) = \mathsf{M}$. It should be the case

that $M'_r(T(w)) \sqsubseteq \mathsf{M}$, because otherwise $\tau|^0_{\mathsf{M}} \trianglelefteq \tau'|^0_{\mathsf{M}}$ would not hold, since observation $\langle w, M_r(w)\rangle$ would belong to $\tau|^0_{\mathsf{M}}$, but no observation involving w would belong to $\tau'|^0_{\mathsf{M}}$.

Assume for contradiction that $M'_r(T(w)) \sqsubset \mathsf{M}$. So, $M'_r(T(w)) = \mathsf{L}$. From (38) and (a1), we get

$$M'(l) \not> 1 \text{ and } M'(m_b) > 0 \tag{39}$$

There exists a memory M'' such that $M'' \models \mathcal{H}_0(E', \mathcal{L}, pgm)$ and:

$$M'|_{\mathsf{L}} = M''|_{\mathsf{L}}, \tag{40}$$
$$M''(m_b) \not> 0, \tag{41}$$
$$M'(m_b) \neq M''(h) \tag{42}$$

Let $\tau'' = trace_{E'}(pgm, M'')$. From (40) and because $T(l) = \mathsf{L}$, we get

$$M'(l) = M''(l). \tag{43}$$

From $M'(l) \not> 1$, we then get

$$M''(l) \not> 1. \tag{44}$$

If $M''_r(T(w)) \neq \mathsf{L}$, then E' does not satisfy 0-BNI($\mathcal{L}$), because (40) and $M'_r(T(w)) = \mathsf{L}$, and thus, observation $\langle w, M'_r(w)\rangle$ is included in $\tau'|^0_{\mathsf{L}}$ but no observation involving w is included in $\tau''|^0_{\mathsf{L}}$. So, $M''_r(T(w)) = \mathsf{L}$. Because E' is an enforcer and due to (44), (39), and (41), we have:
$M'_r(w) = M'_r(w_c) = M'_r(w_b) = M'(m_b)$ and
$M''_r(w) = M''_r(w_c) = M''_r(w_b) = M''(h)$.
From (42), we then have $M'_r(w) \neq M''_r(w)$. So, E' does not satisfy 0-BNI given (40), because $\tau'|^0_{\mathsf{L}}$ includes observation $\langle w, M'_r(w)\rangle$, $\tau''|^0_{\mathsf{L}}$ includes observation $\langle w, M''_r(w)\rangle$, and $M'_r(w) \neq M''_r(w)$. But this is a contradiction. Thus, $M'_r(T(w)) = \mathsf{M}$.

2. $M(l) \not> 1$ and $M(m_b) \not> 0$ (a2)
We have $M_r(T(w)) = \mathsf{H}$. We prove that $M'_r(T(w)) = \mathsf{H}$. If $M'_r(T(w)) = \mathsf{L}$, then we follow the arguments of the above case, where (39) would be $M'(m_b) \not> 0$ and (41) would be $M''(m_b) > 0$, and we are lead to a contradiction.

Assume for contradiction that $M'_r(T(w)) = \mathsf{M}$. There exists M'' such that $M'' \models \mathcal{H}_0(E', \mathcal{L}, pgm)$ and

$$M'|_{\mathsf{M}} = M''|_{\mathsf{M}}, \tag{45}$$
$$M'(h) \neq M''(h). \tag{46}$$

Let $\tau'' = trace_{E'}(pgm, M'')$. From (45), we get

$$M'(l) = M''(l), \tag{47}$$
$$M'(m_b) = M''(m_b). \tag{48}$$

From (38), (a2), (47), and (48), we get

$$M'(l) \not> 1, M''(l) \not> 1, M'(m_b) \not> 0, \text{ and } M''(m_b) \not> 0. \tag{49}$$

It should be the case that $M''_r(T(w)) = \mathsf{M}$, because otherwise E' would not satisfy 0-BNI($\mathcal{L}$), which is a contradiction. Because E' is an enforcer and due to (49), we have:
$M'_r(w) = M'_r(w_c) = M'_r(w_b) = M'(h)$ and
$M''_r(w) = M''_r(w_c) = M''_r(w_b) = M''(h)$.
From (46), we then have $M'_r(w) \neq M''_r(w)$. So, given (45), E' does not satisfy 0-BNI($\mathcal{L}$), which is a contradiction. Thus, $M'_r(T(w)) = \mathsf{H}$.

3. $M(l) > 1$ (a3)
We have $M_r(T(w)) = \mathsf{M}$. We prove that $M'_r(T(w)) = \mathsf{M}$. It should be the case that $M'_r(T(w)) \sqsubseteq \mathsf{M}$, because otherwise $\tau|^0_{\mathsf{M}} \trianglelefteq \tau'|^0_{\mathsf{M}}$ would not hold.

Assume for contradiction that $M'_r(T(w)) \sqsubset \mathsf{M}$. So, $M'_r(T(w)) = \mathsf{L}$. There exists M'' such that

$$M'|_{\mathsf{L}} = M''|_{\mathsf{L}}, \tag{50}$$
$$M'(m_a) \neq M''(m_a) \tag{51}$$

Let $\tau'' = trace_{E'}(C, M'')$. From (50), we get

$$M'(l) = M''(l). \tag{52}$$

From (38) and (a3), we then have

$$M'(l) > 1 \text{ and } M''(l) > 1. \tag{53}$$

If $M''(T(w)) \neq \mathsf{L}$, then E' does not satisfy 0-BNI($\mathcal{L}$), which is a contradiction.
Assume $M''(T(w)) = \mathsf{L}$. Because E' is an enforcer and due to (53), we have
$M'_r(w) = M'_r(w_c) = M'_r(w_a) = M'(m_a)$ and
$M''_r(w) = M''_r(w_c) = M''_r(w_a) = M''(m_a)$.
From (51), we have $M'_r(w) \neq M''_r(w)$. So, given (50), E' does not satisfy 0-BNI($\mathcal{L}$), which is a contradiction. Thus, $M'_r(T(w)) = \mathsf{M}$.

So, for all executions, $T(w)$ produced by 2-*Enf* is 1-precise.

Consider an enforcer E that uses 1-dependent $G^E_{a:=e}$ and 2-*Enf* $\leq^{0,\mathcal{L}}_c E$. We prove that E does not satisfy 0-BNI($\mathcal{L}$). Assume for contradiction that E satisfies 0-BNI($\mathcal{L}$). We examine whether E decides to block the execution of *pgm* for the following exhaustive list of cases: $l > 1$, $l \not> 1 \wedge m_b > 0$, and $l \not> 1 \wedge m_b \not> 0$. From that, we will show that E does not satisfy 0-BNI($\mathcal{L}$), which is a contradiction.

1. $l > 1$:
There exists a conventionally initialized memory M with
$M \models \mathcal{H}_0(2\text{-}\mathit{Enf}, \mathcal{L}, \mathit{pgm})$ and $M(l) > 1$.

Let $\tau' = \mathit{trace}_{2\text{-}\mathit{Enf}}(\mathit{pgm}, M)$. There exists memory M_1 such that $M_1 \models \mathcal{H}_0(E, \mathcal{L}, \mathit{pgm})$, $\rho_1(M, M_1)$, and $\tau_1 = \mathit{trace}_E(\mathit{pgm}, M_1)$. Because 2-*Enf* $\leq^{0,\mathcal{L}}_c E$, we have $\forall \ell \in \mathcal{L}$: $\tau'|^0_\ell \trianglelefteq \tau_1|^0_\ell$. From $\rho_1(M, M_1)$, we have

$$M_1(l) > 1. \tag{54}$$

Because $M(l) > 1$, enforcer 2-*Enf* executes $l := 1$, and thus, $\langle l, 1\rangle \in \tau'|^0_{\mathsf{L}}$. From $\forall \ell$: $\tau'|^0_\ell \trianglelefteq \tau_1|^0_\ell$, we then get $\langle l, 1\rangle \in \tau_1|^0_{\mathsf{L}}$. Thus, $\langle l := 1, M_r\rangle \to \langle \mathbf{stop}, M_s\rangle$ should be a subtrace of τ_1. Thus, we have

$$M_r(G^E_{l:=1}) = \mathit{true}. \tag{55}$$

Because *pgm* is an anchor-tailed command, we can have $V_{l:=1} = \{l, m, w\}$. Because E uses 1-dependent $G^E_{l:=1}$, we get that

$$M_r(G^E_{l:=1}) = f(M_r(T(l)), M_r(T(m)), M_r(T(w))). \tag{56}$$

Because, for all executions, $T(w)$ produced by 2-*Enf* is 1-precise, and because 2-*Enf* $\leq^{0,\mathcal{L}}_c E$, E satisfies 0-BNI($\mathcal{L}$), and $\rho_1(M, M_1)$, we get

$$M_r(T(l)) = \mathsf{L}, M_r(T(m)) = \mathsf{M}, M_r(T(w)) = \mathsf{M}. \tag{57}$$

From (55) and (56), we then get

$$f(\mathsf{L}, \mathsf{M}, \mathsf{M}) = \mathit{true}. \tag{58}$$

2. $l \not> 1$ and $m_b > 0$:
There exists a conventionally initialized memory M' with $M' \models \mathcal{H}_0(2\text{-}\mathit{Enf}, \mathcal{L}, \mathit{pgm})$ $M'(l) \not> 1$ and $M'(m_b) > 0$.

Let $\tau' = \mathit{trace}_{2\text{-}\mathit{Enf}}(\mathit{pgm}, M')$. There exists M_2 such that $M_2 \models \mathcal{H}_0(E, \mathcal{L}, \mathit{pgm})$, $\rho_1(M', M_2)$, and $\tau_2 = \mathit{trace}_E(\mathit{pgm}, M_2)$. Because $2\text{-}\mathit{Enf} \leq_c^{0,\mathcal{L}} E$, we have that $\forall \ell \in \mathcal{L}$: $\tau'|_\ell^0 \trianglelefteq \tau_2|_\ell^0$. From $\rho_1(M', M_2)$, we have

$$M_2(l) \not> 1 \ \wedge \ M_2(m_b) > 0. \tag{59}$$

Because $M'(l) \not> 1$, and $M'(m_b) > 0$, enforcer $2\text{-}\mathit{Enf}$ executes $m := w$, and thus, $\langle m, \nu\rangle \in \tau'|_{\mathsf{M}}^0$. From $\forall \ell$: $\tau'|_\ell^0 \trianglelefteq \tau_2|_\ell^0$, we then get $\langle m, \nu\rangle \in \tau_2|_{\mathsf{M}}^0$. Thus, $\langle m := w; l := 1, M_m\rangle \to \langle l := 1, M_r\rangle$ should be a subtrace of τ_2. Because pgm is an anchor-tailed command, we can have $V_{l:=1} = \{l, m, w\}$. Because E uses 1-dependent $G_{l:=1}^E$, we get that

$$M_r(G_{l:=1}^E) = f(M_r(T(l)), M_r(T(m)), M_r(T(w))). \tag{60}$$

Because, for all executions, $T(w)$ produced by $2\text{-}\mathit{Enf}$ is 1-precise, and because $2\text{-}\mathit{Enf} \leq_c^{0,\mathcal{L}} E$, E satisfies 0-BNI($\mathcal{L}$), and $\rho_1(M', M_2)$, we get

$$M_r(T(l)) = \mathsf{L}, M_r(T(m)) = \mathsf{M}, M_r(T(w)) = \mathsf{M}. \tag{61}$$

So, $M_r(G_{l:=1}^E) = f(\mathsf{L}, \mathsf{M}, \mathsf{M}) = \mathit{true}$, due to (58). So, E executes $l := 1$, and thus, $\langle l, 1\rangle \in \tau_2|_{\mathsf{L}}^0$. (c1)

3. $l \not> 1$ and $m_b \not> 0$:
There exists a conventionally initialized memory M'' with $M'' \models \mathcal{H}_0(2\text{-}\mathit{Enf}, \mathcal{L}, \mathit{pgm})$, $\mathit{dom}(M'') = \mathit{dom}(M')$, $M'|_{\mathsf{L}} = M''|_{\mathsf{L}}$, $M''(l) \not> 1$, and $M''(m_b) \not> 0$.

Let $\tau'' = \mathit{trace}_{2\text{-}\mathit{Enf}}(\mathit{pgm}, M'')$. There exists M_3 such that $M_3 \models \mathcal{H}_0(E, \mathcal{L}, \mathit{pgm})$, $\rho_1(M'', M_3)$, and $\tau_3 = \mathit{trace}_E(\mathit{pgm}, M_3)$. Because $2\text{-}\mathit{Enf} \leq_c^{0,\mathcal{L}} E$, we have that $\forall \ell \in \mathcal{L}$: $\tau''|_\ell^0 \trianglelefteq \tau_3|_\ell^0$. From $\rho_1(M'', M_3)$, we have

$$M_3(l) \not> 1 \ \wedge \ M_3(m_b) \not> 0. \tag{62}$$

From $dom(M'') = dom(M')$, $M'|_{\mathsf{L}} = M''|_{\mathsf{L}}$, $\rho_1(M'', M_3)$, $c(M'')$, $\rho_1(M', M_2)$, $c(M')$, we then get $M_2|_{\mathsf{L}} = M_3|_{\mathsf{L}}$. Enforcer 2-*Enf* executes $w := w_c$, and thus, $\langle w, \nu \rangle \in \tau''|^0_{\mathsf{H}}$. From $\forall \ell \colon \tau''|^0_\ell \trianglelefteq \tau_3|^0_\ell$, we then get $\langle w, \nu \rangle \in \tau_3|^0_{\mathsf{H}}$. Thus, $\langle w := w_c; m := w; l := 1, M_w \rangle \to \langle m := w; l := 1, M_m \rangle$ should be a subtrace of τ_3. We examine two cases: τ_3 either executes $m := w$ or not.

3.1. τ_3 does not execute $m := w$.
So, $l := 1$ is not executed either. (c2)
Thus, $\tau_2|^0_{\mathsf{L}} \neq \tau_3|^0_{\mathsf{L}}$, due to (c1) and (c2).
From $M_2|_{\mathsf{L}} = M_3|_{\mathsf{L}}$, we then get that E does not satisfy 0-BNI($\mathcal{L}$).

3.2. τ_3 executes $m := w$.
Then h is leaked to m. There exists M_4 such that $M_4 \models \mathcal{H}_0(E, \mathcal{L}, pgm)$ and

$$M_3|_{\mathsf{M}} = M_4|_{\mathsf{M}}, \tag{63}$$
$$M_3(h) \neq M_4(h). \tag{64}$$

From (63) and (62), we get

$$M_3(l) = M_4(l) \not> 1, \tag{65}$$
$$M_3(m_b) = M_4(m_b) \not> 0. \tag{66}$$

Let $\tau_4 = trace_E(pgm, M_4)$. If τ_4 does not execute $m := w$, then $\tau_3|_{\mathsf{M}} \neq \tau_4|_{\mathsf{M}}$. If τ_4 executes $m := w$, then (64) implies $\tau_3|_{\mathsf{M}} \neq \tau_4|_{\mathsf{M}}$, because in both traces τ_3 and τ_4 the value of m equals the value of h. So, in both cases, E does not satisfy 0-BNI($\mathcal{L}$).

□

Familiar Two-level Lattice

Theorem 8. *Enforcer $E_{\mathsf{H,L}}$ uses 1-dependent $G_{a:=e}$, satisfies 0-BNI($\mathcal{L}_2$), and satisfies 2-Enf $<^{0,\mathcal{L}_2}_c$ $E_{\mathsf{H,L}}$.*

Proof. Lemma 24 gives that $E_{\mathsf{H,L}}$ is an enforcer and uses 1-dependent $G_{a:=e}$. Lemma 21 gives that $E_{\mathsf{H,L}}$ satisfies 0-BNI($\mathcal{L}_2$). Lemma 22 gives that 2-*Enf* $<^{0,\mathcal{L}_2}_c$ $E_{\mathsf{H,L}}$ holds. □

Lemma 21. *$E_{\mathsf{H,L}}$ satisfies 0-BNI($\mathcal{L}_2$).*

$$(\text{AsgnA})\ \frac{v = M(e) \qquad G_{a:=e} \qquad \ell = M(\lfloor cc \rfloor) \sqcup M(bc)}{\langle a := e, M\rangle \to \langle \textbf{stop}, M[a \mapsto v, bc \mapsto \ell]\rangle}$$

$$(\text{AsgnAFail})\ \frac{v = M(e) \qquad \neg G_{a:=e} \qquad \ell = M(\lfloor cc \rfloor) \sqcup M(bc)}{\langle a := e, M\rangle \to \langle \textbf{block}, M[bc \mapsto \ell]\rangle}$$

$$(\text{AsgnF})\ \frac{\nu_0 = M(e) \qquad \nu_1 = M(T(e)) \sqcup M(\lfloor cc \rfloor) \sqcup M(bc)}{\langle w := e, M\rangle \to \langle \textbf{stop}, M[w \mapsto \nu_0, T(w) \mapsto \nu_1]\rangle}$$

$$U(M, W) \triangleq M[\,\forall w{\in}W\colon T(w) \mapsto T(w) \sqcup M(\lfloor cc \rfloor) \sqcup M(bc)]$$

Figure 11: Modified rules for $E_{\mathsf{H,L}}$

Proof. We retrieve $E_{\mathsf{H,L}}$ from ∞-*Enf* by replacing rules for assignments and function U used in **exit** with the corresponding definitions in Figure 11. Because BNI+(∞-*Enf*,$\mathcal{L}_2$,C) holds, then BNI+($E_{\mathsf{H,L}}$,$\mathcal{L}_2$,C) holds where C is any command different from assignment and **exit**, provided all Lemmata used to prove Lemma 1 hold for rules in Figure 11. In particular, Lemmata 2, 3, 4, 7, 8, 5, and 6 still hold. Now, it suffices to prove BNI+($E_{\mathsf{H,L}}$,$\mathcal{L}_2$,C) where C is an assignment or **exit**.

For $E_{\mathsf{H,L}}$, the domain of memories contain only variables and tags of these variables. So, in the definition of BNI+, projections $M|_\ell$ and $\tau|_\ell^k$ equal projection $M|_\ell^0$ and $\tau|_\ell^0$, correspondingly. Also, $mon(M)$ is trivially true for any such memory So, the definition on BNI+ becomes: For all $\ell \in \mathcal{L}_2, M, M'$ if

$$\begin{aligned}
&M|_\ell^0 = M'|_\ell^0,\\
&M(cc) = M'(cc)\\
&\tau = \mathit{trace}_{E_{\mathsf{H,L}}}(C, M) = \langle C, M\rangle \xrightarrow{*} \langle C_t, M_t\rangle,\\
&\tau' = \mathit{trace}_{E_{\mathsf{H,L}}}(C, M') = \langle C, M'\rangle \xrightarrow{*} \langle C'_t, M'_t\rangle
\end{aligned}$$

where C_t and C'_t are terminations (i.e., **stop** or **block**), then:

c1 If C_t and C'_t are both **stop**, then $\tau|_\ell^0 =_{obs} \tau'|_\ell^0$, $M_t|_\ell^0 = M'_t|_\ell^0$, and $M_t(cc) = M'_t(cc)$.

c2 If C_t or C'_t is **block**, then $\tau|_\ell^0 =_{obs} \tau'|_\ell^0$.

c3 If C_t is **stop**, C'_t is **block**, and $M'_t(bc) \not\sqsubseteq \ell$, then $M_t(bc) \not\sqsubseteq \ell$.

c4 If C_t is **stop**, C'_t is **block**, $\langle C'_{tp}, M'_{tp}\rangle \to \langle C'_t, M'_t\rangle$ are the last two states of τ', and $M'_{tp}(\lfloor cc\rfloor) \sqsubseteq \ell$, then there exists $\langle C'', M''\rangle \in \tau$, with $C'' = C'_{tp}$ and $M''|_\ell = M'_{tp}|_\ell$.

Because $\ell \in \mathcal{L}_2$,we have that ℓ is either L or H. If $\ell = \mathsf{H}$, then BNI+ is trivially satisfied, because hypothesis $M|^0_\ell = M'|^0_\ell$ implies $M = M'$. Thus, we prove BNI+ for

$$\ell = \mathsf{L}. \tag{67}$$

1. C is $a := e$:
We prove that $M(T(a)) = M'(T(a))$. If $M(T(a)) = \mathsf{L}$, then $M|_\ell = M'|_\ell$ gives $M'(T(a)) = \mathsf{L}$. If $M(T(a)) = \mathsf{H}$, then $M|_\ell = M'|_\ell$ gives $M'(T(a)) = \mathsf{H}$. So, $M(T(a)) = M'(T(a))$.

1.1. $M(T(a)) \sqsubseteq \ell$
We first prove that the command is executed normally in both memories or blocked in both memories. W.l.o.g, assume that the command is executed normally in M. That is, $M(T(e)) \sqcup M(\lfloor cc\rfloor) \sqcup M(bc) \sqsubseteq M(T(a))$ holds. This implies that $M(T(e)) \sqsubseteq \ell$, $M(\lfloor cc\rfloor) \sqsubseteq \ell$, and $M(bc) \sqsubseteq \ell$. Because $M|_\ell = M'|_\ell$, we get $M(cc) = M'(cc)$ and $M(bc) = M'(bc)$. From $M(T(e)) \sqsubseteq \ell$ and $M|_\ell = M'|_\ell$, we then get $M'(T(e)) \sqsubseteq \ell$. From (67), we then have $M(T(e)) = M'(T(e)) = \mathsf{L}$. Thus, $M'(T(e)) \sqcup M'(\lfloor cc\rfloor) \sqcup M'(bc) \sqsubseteq M'(T(a))$ holds. So, in both cases the command is executed normally. Thus, the command is executed normally in both memories or blocked in both memories. So, $c3$ and $c4$ are trivially true.

1.1.1. The command is executed normally in both memories.
$c2$ is trivially true.
We prove $c1$. We have:
$\tau = \langle a := e, \mathcal{M}\rangle \to \langle \textbf{stop}, M[a \mapsto M(e), bc \mapsto \ell_g]\rangle$
$\tau' = \langle a := e, \mathcal{M}'\rangle \to \langle \textbf{stop}, M'[a \mapsto M'(e), bc \mapsto \ell'_g]\rangle$,
where $\ell_g = M(\lfloor cc\rfloor) \sqcup M(bc)$ and $\ell'_g = M'(\lfloor cc\rfloor) \sqcup M'(bc)$. We have $\tau|_\ell = \tau'|_\ell = \langle a, M(e)\rangle$, because $M(e) = M'(e)$. Also, $\ell_g = \ell'_g$. So, $M_t(bc) = M'_t(bc)$. Because $M_t(cc) = M(cc) = M'(cc) = M'_t(cc)$, $M_t(cc) = M'_t(cc)$ holds. Because $M_t(a) = M'_t(a)$, $M_t(bc) = M'_t(bc)$, and $M_t(cc) = M'_t(cc)$, we get $M_t|_\ell = M'_t|_\ell$. Thus $c1$ holds.

1.1.2. The command is blocked in both memories.

$\tau = \langle a := e, \mathcal{M}\rangle \rightarrow \langle \mathbf{block}, M[bc \mapsto \ell_g]\rangle$

$\tau' = \langle a := e, \mathcal{M}'\rangle \rightarrow \langle \mathbf{block}, M'[bc \mapsto \ell_g']\rangle$.

So, $\tau|_\ell =_{obs} \epsilon$ and $\tau'|_\ell =_{obs} \epsilon$. Thus $c2$ holds. $c1$ is trivially true.

1.2. $M(T(a)) \not\sqsubseteq \ell$

We have $\tau|_\ell =_{obs} \epsilon$ and $\tau'|_\ell =_{obs} \epsilon$. Thus $c2$ holds.

We prove $c1$. Assume $C_t = C_t' = \mathbf{stop}$. Because $M(T(a)), M'(T(a)) \not\sqsubseteq \ell$, M_t and M_t' do not need to agree on a. Assume $M_t(bc) \sqsubseteq \ell$. So, $M(T^2(e)) \sqcup M(\lfloor cc \rfloor) \sqcup M(bc) \sqsubseteq \ell$. Thus, $M(T^2(e)) \sqsubseteq \ell$, $M(\lfloor cc \rfloor) \sqsubseteq \ell$, and $M(bc) \sqsubseteq \ell$. From $mon(M)$ and $M(T^2(e)) \sqsubseteq \ell$, we get $M(T^3(e)) \sqsubseteq \ell$. From $M|_\ell = M'|_\ell$, $M(T^3(e)) \sqsubseteq \ell$, $M(\lfloor cc \rfloor) \sqsubseteq \ell$, $M(bc) \sqsubseteq \ell$, and Lemma 5, we get: $M(T^2(e)) = M'(T^2(e))$, $M(\lfloor cc \rfloor) = M'(\lfloor cc \rfloor)$, and $M(bc) = M'(bc)$. So, $M_t(bc) = M_t'(bc)$. From $M(cc) = M'(cc)$, we get $M(cc) = M'(cc)$. From $M_t(cc) = M(cc)$ and $M_t'(cc) = M'(cc)$, we then get $M_t(cc) = M_t'(cc)$. Thus, $M_t|_\ell = M_t'|_\ell$ and $M_t(cc) = M_t'(cc)$. Thus $c1$ holds.

We prove $c3$. Assume C_t is $\mathbf{stop}$, C_t' is $\mathbf{block}$, and $M_t'(bc) \not\sqsubseteq \ell$. We prove $M_t(bc) \not\sqsubseteq \ell$. We prove the contrapositive. Assume $M_t(bc) \sqsubseteq \ell$, then following the same arguments as above, we get $M_t(bc) = M_t'(bc)$, and thus, $M_t'(bc) \sqsubseteq \ell$, as wanted.

We prove $c4$. Assume C_t is $\mathbf{stop}$, C_t' is $\mathbf{block}$, $\langle C_{tp}', M_{tp}'\rangle \rightarrow \langle C_t', M_t'\rangle$ are the last two states of τ', and $M_{tp}'(\lfloor cc \rfloor) \sqsubseteq \ell$. So, $M_{tp}' = M'$ and $C_{tp}' = a := e$. We have that $\langle C'', M''\rangle = \langle a := e, M\rangle$, which satisfies $C'' = C_{tp}'$ and $M''|_\ell = M_{tp}'|_\ell$. Thus $c4$ holds.

2. C is $w := e$

$\tau = \langle w := e, M\rangle \rightarrow \langle \mathbf{stop}, M_t\rangle$

$\tau' = \langle w := e, M'\rangle \rightarrow \langle \mathbf{stop}, M_t'\rangle$

$c2$, $c3$, and $c4$ are trivially true.

We prove $c1$. $M_t(cc) = M_t'(cc)$ holds due to $M(cc) = M'(cc)$, $M_t(cc) = M(cc)$, and $M_t'(cc) = M'(cc)$.

2.1. $M_t(T(w)) \sqsubseteq \ell$

Then $M(T(e)) \sqsubseteq \ell$, $M(\lfloor cc \rfloor) \sqsubseteq \ell$, and $M(bc) \sqsubseteq \ell$. From, $M|_\ell = M'|_\ell$ and (67), we then get $M(e) = M'(e)$, $M(T(e)) = M'(T(e))$, $M(cc) = M'(cc)$, and $M(bc) = M'(bc)$. So, $M_t(w) = M_t'(w)$ and

$M_t(T(w)) = M'_t(T(w))$. Thus, $M_t|_\ell = M'_t|_\ell$ and $\tau|_\ell = \tau'|_\ell$. Thus $c1$ holds.

2.2. $M_t(T(w)) \not\sqsubseteq \ell$
By symmetry of preceding case, $M'_t(T(w)) \not\sqsubseteq \ell$. So, $\tau|_\ell =_{obs} \epsilon$ and $\tau'|_\ell =_{obs} \epsilon$. Also $M_t|_\ell = M'_t|_\ell$. Thus $c1$ holds.

3. **exit**
$\tau = \langle \textbf{exit}, M\rangle \to \langle \textbf{stop}, M_t\rangle$
$\tau' = \langle \textbf{exit}, M'\rangle \to \langle \textbf{stop}, M'_t\rangle$.
$c2$, $c3$, $c4$ are trivially true.

We prove $c1$. We have $\tau|_\ell =_{obs} \epsilon$ and $\tau'|_\ell =_{obs} \epsilon$. So, we need to prove $M_t|_\ell = M_t|_\ell$ and $M_t(cc) = M'_t(cc)$. From $M(cc) = M'(cc)$, we get $M(cc) = M'(cc)$. Because $M_t(cc) = M(cc).pop$ and $M'_t(cc) = M'(cc).pop$, we then get $M_t(cc) = M'_t(cc)$. We now prove $M_t|_\ell = M_t|_\ell$.

3.1. $M_t(\lfloor cc\rfloor) \sqcup M_t(bc) \not\sqsubseteq \ell$ and $M(cc).top.A \neq \emptyset$.
Because $M(cc).top.A \neq \emptyset$, we have $M_t(bc) = M(bc) \sqcup M(\lfloor cc\rfloor)$. We have $M_t(\lfloor cc\rfloor) \sqsubseteq M(\lfloor cc\rfloor)$. We get $M_t(\lfloor cc\rfloor) \sqcup M(bc) \sqsubseteq M(\lfloor cc\rfloor) \sqcup M(bc)$, which becomes $M_t(\lfloor cc\rfloor) \sqcup M(bc) \sqcup M(\lfloor cc\rfloor) \sqsubseteq M(\lfloor cc\rfloor) \sqcup M(bc) \sqcup M(\lfloor cc\rfloor)$, which becomes $M_t(\lfloor cc\rfloor) \sqcup M_t(bc) \sqsubseteq M(\lfloor cc\rfloor) \sqcup M(bc)$. So, $M(\lfloor cc\rfloor) \sqcup M(bc) \not\sqsubseteq \ell$. So, $M_t(bc) \not\sqsubseteq \ell$. Because $M|_\ell = M'|_\ell$, we also get $M'(\lfloor cc\rfloor) \sqcup M'(bc) \not\sqsubseteq \ell$.

$M(cc).top.A \neq \emptyset$ and $M(cc) = M'(cc)$ give $M'(cc).top.A \neq \emptyset$, too. So, $M'_t(bc) = M'(bc) \sqcup M'(\lfloor cc\rfloor)$. Thus, $M'_t(bc) \not\sqsubseteq \ell$.

From $M_t(bc) \not\sqsubseteq \ell$ and $M'_t(bc) \not\sqsubseteq \ell$, we get $M_t(\lfloor cc\rfloor) \sqcup M_t(bc) \not\sqsubseteq \ell$ and $M'_t(\lfloor cc\rfloor) \sqcup M'_t(bc) \not\sqsubseteq \ell$.

Only variables in W change their labels. Let $x \in M(cc).top.W$. Because $M(\lfloor cc\rfloor) \sqcup M(bc) \not\sqsubseteq \ell$, we have $M_t(T(x)) \not\sqsubseteq \ell$. Because $M(cc) = M'(cc)$, we get $x \in M'(cc).top.W$, too. Because $M'(\lfloor cc\rfloor) \sqcup M'(bc) \not\sqsubseteq \ell$, we have $M'_t(T(x)) \not\sqsubseteq \ell$. So, $M_t|_\ell = M'_t|_\ell$. Thus $c1$ holds.

3.2. $M_t(\lfloor cc\rfloor) \sqcup M_t(bc) \not\sqsubseteq \ell$ and $M(cc).top.A = \emptyset$.
Because $M(cc).top.A = \emptyset$, we have $M_t(bc) = M(bc)$. We have $M_t(\lfloor cc\rfloor) \sqsubseteq M(\lfloor cc\rfloor)$. We get $M_t(\lfloor cc\rfloor) \sqcup M(bc) \sqsubseteq M(\lfloor cc\rfloor) \sqcup M(bc)$, which becomes $M_t(\lfloor cc\rfloor) \sqcup M_t(bc) \sqsubseteq M(\lfloor cc\rfloor) \sqcup M(bc)$. So, $M(\lfloor cc\rfloor) \sqcup M(bc) \not\sqsubseteq \ell$. Because $M|_\ell = M'|_\ell$, we also get $M'(\lfloor cc\rfloor) \sqcup M'(bc) \not\sqsubseteq \ell$.

$M(cc).top.A = \emptyset$ and $M(cc) = M'(cc)$ give $M'(cc).top.A = \emptyset$. So, $M'_t(bc) = M'(bc)$.

Assume $M_t(bc) \sqsubseteq \ell$. Then $M(bc) \sqsubseteq \ell$. From $M|_\ell = M'|_\ell$, we then get $M(bc) = M'(bc)$. Thus, $M_t(bc) = M'_t(bc)$.

We prove $M'_t(\lfloor cc \rfloor) \sqcup M'_t(bc) \not\sqsubseteq \ell$. Assume for contradiction that $M'_t(\lfloor cc \rfloor) \sqcup M'_t(bc) \sqsubseteq \ell$. Then $M'_t(bc) \sqsubseteq \ell$. Following the same arguments as above, we get $M_t(bc) = M'_t(bc)$. Because $M_t(cc) = M'_t(cc)$, we then get $M_t(bc) \sqcup M_t(cc) = M'_t(bc) \sqcup M'_t(cc)$, which is a contradiction. So, $M'_t(\lfloor cc \rfloor) \sqcup M'_t(bc) \not\sqsubseteq \ell$.

Only variables in V change their labels. Let $x \in M(cc).top.W$. Because $M(\lfloor cc \rfloor) \sqcup M(bc) \not\sqsubseteq \ell$, we have $M_t(T(x)) \not\sqsubseteq \ell$. Because $M(cc) = M'(cc)$, we get $x \in M'(cc).top.W$, too. Because $M'(\lfloor cc \rfloor) \sqcup M'(bc) \not\sqsubseteq \ell$, we have $M'_t(T(x)) \not\sqsubseteq \ell$. So, $M_t|_\ell = M'_t|_\ell$. Thus $c1$ holds.

3.3. $M_t(\lfloor cc \rfloor) \sqcup M_t(bc) \sqsubseteq \ell$

So, $M_t(bc) \sqsubseteq \ell$. From Lemma 7, we get $M(bc) \sqsubseteq \ell$. From $M|_\ell = M'|_\ell$ and $M(bc) \sqsubseteq \ell$, we also get $M(bc) = M'(bc)$. From $M(cc) = M'(cc)$, we then get $M_t(bc) = M'_t(bc)$.

- Let $M(\lfloor cc \rfloor) \sqcup M(bc) \sqsubseteq \ell$.
  Let $x \in M(cc).top.W$. Because $M(cc) = M'(cc)$, we get $x \in M'(cc).top.W$. We have:
  $M_t(T(x)) \sqsubseteq \ell \Rightarrow M(T(x)) \sqsubseteq \ell \Rightarrow M(x) = M'(x)$ and $M'(T(x)) \sqsubseteq \ell$.
  Because $M(cc) = M'(cc)$ and $M(bc) = M'(bc)$, we then have $M_t(x) = M'_t(x)$ and $M'_t(T(x)) \sqsubseteq \ell$. Thus, $M_t|_\ell = M'_t|_\ell$. Thus $c1$ holds.
- Let $M(\lfloor cc \rfloor) \sqcup M(bc) \not\sqsubseteq \ell$.
  So, $M'(\lfloor cc \rfloor) \sqcup M'(bc) \not\sqsubseteq \ell$. Let $x \in M(cc).top.W$. So, $M_t(T(x)) \not\sqsubseteq \ell$. Because $M(cc) = M'(cc)$, we get $x \in M'(cc).top.V$, and thus $M'_t(T(x)) \not\sqsubseteq \ell$. Thus, $M_t|_\ell = M'_t|_\ell$. Thus $c1$ holds.

Case *4.3.2.* in the proof of Lemma 1 mentions AsgnA.
We reexamine this case when rule AsgnA in Figure 11 is instead used:

– $M_{tp}(bc) \sqsubseteq \ell$ and $M_{tp}(cc) \sqsubseteq \ell$
  From IH[$c4$] on C_1, there exists $\langle C_{1tp}, M'_1 \rangle \in \tau'_1$ such that $M_{tp}|_\ell = M'_1|_\ell$. So, $M_{tp}(bc) = M'_1(bc)$ and $M_{tp}(cc) = M'_1(cc)$. Because τ_1 is blocked, we have $M_{tp}(T(e)) \sqcup M_{tp}(\lfloor cc \rfloor) \sqcup M_{tp}(bc) \not\sqsubseteq M_{tp}(T(a))$. Since the inequality is satisfied in τ'_1, it means that the value of $T(e)$ is different in M'_1 and M_{tp}. But this contradicts $M_{tp}|_\ell = M'_1|_\ell$. Indeed, $M_{tp}|_\ell = M'_1|_\ell$ implies

that $M_1'(T(e))$ and $M_{tp}(T(e))$ should either be both L or both H. Thus, this case is no longer possible once a two-level lattice is considered.

Thus, BNI+($E_{\mathsf{H,L}}$,$\mathcal{L}_2$,C) holds. BNI+ implies 0-BNI. So, $E_{\mathsf{H,L}}$ satisfies 0-BNI. $\square$

Lemma 22. 2-*Enf* $<_c^{0,\mathcal{L}_2}$ $E_{\mathsf{H,L}}$

Proof. We first prove 2-*Enf* $\leq_c^{0,\mathcal{L}_2}$ $E_{\mathsf{H,L}}$. Consider conventionally initialized memory M with $M \models \mathcal{H}_0(2\text{-}\mathit{Enf}, \mathcal{L}_2, C)$. Consider memory M' with $M' \models \mathcal{H}_0(E_{\mathsf{H,L}}, \mathcal{L}_2, C)$ and $\rho_1(M, M')$. Assume $\tau' = \mathit{trace}_{E_{\mathsf{H,L}}}(C, M')$ and $\tau = \mathit{trace}_{2\text{-}\mathit{Enf}}(C, M)$. By definition, we have $M'(cc) = M(cc) = \epsilon$ and $M'(bc) = M(bc) = \perp$.

We write $M' \sqsubseteq_e M$ iff

1. $\forall x$: $M'(x) = M(x)$,
2. $\forall a$: $M'(T(a)) = M(T(a))$,
3. $\forall w$: $M'(T(w)) \sqsubseteq M(T(w))$,
4. $M'(bc) \sqsubseteq M(bc)$,
5. $\forall i \geq 0$: $M'(\lfloor cc.pop^i \rfloor) \sqsubseteq M(\lfloor cc.pop^i \rfloor) \wedge$
   $M'(cc.pop^i.top.A) = M(cc.pop^i.top.A) \wedge$
   $M'(cc.pop^i.top.W) = M(cc.pop^i.top.W)$

where $cc.pop^i$ pops the top i elements from cc and $cc.pop^0 = cc$.

So, $M' \sqsubseteq_e M$ holds. By induction on the number of steps in τ and Lemma 23, we get that $\forall \ell \in \mathcal{L}_2$: $\tau|_\ell^0 \trianglelefteq \tau'|_\ell^0$. So, 2-*Enf* $\leq_c^{0,\mathcal{L}_2}$ $E_{\mathsf{H,L}}$.

Now, we prove $E_{\mathsf{H,L}} \not\leq_c^{0,\mathcal{L}_2}$ 2-*Enf*. Consider program *pgm*:

```
if h > 0
   w := h
else
   w := l
end;
h := w;
l := 1
```

For every execution under 2-*Enf* we have:

- w is associated with $\langle \mathsf{H}, \mathsf{H} \rangle$,
- bc is H when reaching $l := 1$,
- so $l := 1$ is blocked.

For every execution under $E_{\mathsf{H},\mathsf{L}}$ we have:

- w is associated with H,
- bc is L when reaching $l := 1$,
- so $l := 1$ is allowed.

Thus, $E_{\mathsf{H},\mathsf{L}}$ produces observation $\langle l, 1 \rangle$, but 2-*Enf* does not. So, $E_{\mathsf{H},\mathsf{L}} \not\leq_c^{0,\mathcal{L}_2}$ 2-*Enf*. From 2-*Enf* $\leq_c^{0,\mathcal{L}_2} E_{\mathsf{H},\mathsf{L}}$, we then have 2-*Enf* $<_c^{0,\mathcal{L}_2} E_{\mathsf{H},\mathsf{L}}$. Notice that the same program *pgm* works for 2-*Eopt*, too. And thus we get 2-*Eopt* $<_c^{0,\mathcal{L}_2} E_{\mathsf{H},\mathsf{L}}$. □

Lemma 23. *If $\langle C_1, M_1 \rangle \rightarrow \langle C_2, M_2 \rangle$ under 2-Enf, and $\langle C_1, M_1' \rangle \rightarrow \langle C_2', M_2' \rangle$ under $E_{\mathsf{H},\mathsf{L}}$, and $M_1' \sqsubseteq_e M_1$, then $C_2 = C_2'$ and $M_2' \sqsubseteq_e M_2$ hold or $C_2 =$ **block** holds.*

Proof. We use induction on C_1. Assume $C_2 \neq$ **block**. We prove $C_2 = C_2'$ and $M_2' \sqsubseteq_e M_2$.

1. C_1 is $a := e$.
   We have $C_2 =$ **stop**. Then $M_1(T(e)) \sqcup M_1(\lfloor cc \rfloor) \sqcup M_1(bc) \sqsubseteq M_1(T(a))$ holds. Due to $M_1' \sqsubseteq_e M_1$, we then get $M_1'(T(e)) \sqcup M_1'(\lfloor cc \rfloor) \sqcup M_1'(bc) \sqsubseteq M_1'(T(a))$. So, $C_2' =$ **stop**. Also, $M_2(a) = M_1(e) = M_1'(e) = M_2'(a)$. Because $M_2'(bc) = M_1'(\lfloor cc \rfloor) \sqcup M_1'(bc)$ and $M_2(bc) = M_1(T(e)) \sqcup M_1(\lfloor cc \rfloor) \sqcup M_1(bc)$, hypothesis $M_1' \sqsubseteq_e M_1$ gives $M_2'(bc) \sqsubseteq M_2(bc)$. So, $M_2' \sqsubseteq_e M_2$.

2. C_1 is $w := e$.
   We have $C_2 = C_2' =$ **stop**. Hypothesis $M_1' \sqsubseteq_e M_1$ implies $M_1(e) = M_1'(e)$, and thus, $M_2(w) = M_2'(w)$. Because $M_2'(T(w)) = M_1'(T(e)) \sqcup M_1'(\lfloor cc \rfloor) \sqcup M_1'(bc)$ and $M_2(T(w)) = M_1(T(e)) \sqcup M_1(\lfloor cc \rfloor) \sqcup M_1(bc)$, hypothesis $M_1' \sqsubseteq_e M_1$ gives $M_2'(T(w)) \sqsubseteq M_2(T(w))$. So, $M_2' \sqsubseteq_e M_2$.

3. C_1 is **exit**
   We have $C_2 = C_2' =$ **stop**. Because $M_2(cc) = M_1(cc).pop$ and $M_2'(cc) = M_1'(cc).pop$, hypothesis $M_1' \sqsubseteq_e M_1$ gives $\forall i$: $M_2'(\lfloor cc.pop^i \rfloor) \sqsubseteq M_2(\lfloor cc.pop^i \rfloor)$.

Hypothesis $M'_1 \sqsubseteq_e M_1$ also gives $M'_1(cc) \sqsubseteq M_1(cc)$, $M'_1(cc.top.A) = M_1(cc.top.A)$, and $M'_1(bc) \sqsubseteq M_1(bc)$, and thus, we get $M'_2(bc) \sqsubseteq M_2(bc)$. From $M'_1 \sqsubseteq_e M_1$, we also get $M'_1(cc.top.W) = M_1(cc.top.W)$. Because, for all $w \in W$, we have $M'_2(T(w)) = M'_1(T(w)) \sqcup M'_1(\lfloor cc \rfloor) \sqcup M'_1(bc)$ and $M_2(T(w)) = M_1(T(w)) \sqcup M_1(\lfloor cc \rfloor) \sqcup M_1(bc)$, hypothesis $M'_1 \sqsubseteq_e M_1$ gives $M'_2(T(w)) \sqsubseteq M_2(T(w))$. So, $M'_2 \sqsubseteq_e M_2$.

2-*Enf* and $E_{\mathsf{H,L}}$ use the same rules for other commands, so $M'_2 \sqsubseteq_e M_2$ follows easily. □

Lemma 24. *$E_{\mathsf{H,L}}$ is an enforcer and uses 1-dependent $G_{a:=e}$.*

Proof. It is easy to prove that $E_{\mathsf{H,L}}$ is an enforcer on R and satisfies restrictions (E1), (E2), and (E3) by induction on the operational semantics rules. We omit the details.

We now prove that $E_{\mathsf{H,L}}$ uses 1-dependent $G_{a:=e}$. Consider an assignment $a := e$ in an anchor-tailed command $C; C'$, where C does not involve any assignment to anchor variable and C' is a sequence of assignments to anchor variables. From rule (AsgnA) of $E_{\mathsf{H,L}}$ we get that $G_{a:=e}$ is $M(T(e)) \sqcup M(\lfloor cc \rfloor) \sqcup M(bc) \sqsubseteq M(T(a))$. Because $a := e$ is in an anchor-tailed command, we get that $a := e$ is not encapsulated in any conditional command. So, $M(cc) = \epsilon$. Because C does not involve any assignment to anchor variable, bc is $\bot$ when execution reaches C'. While executing C', cc remains ϵ, and thus, from rule (AsgnA) of $E_{\mathsf{H,L}}$, we get that bc remains $\bot$. So, $M(bc) = \bot$. Thus, $G_{a:=e}$ is $M(T(e)) \sqsubseteq M(T(a))$. So, $E_{\mathsf{H,L}}$ uses 1-dependent $G_{a:=e}$. □

References

[1] A. Askarov, S. Chong, and H. Mantel. Hybrid monitors for concurrent noninterference. In *Proceedings of the 28th IEEE Computer Security Foundations Symposium*, pages 137–151, July 2015.

[2] A. Askarov, S. Hunt, A. Sabelfeld, and D. Sands. Termination-insensitive noninterference leaks more than just a bit. In *Proceedings of the 13th European Symposium on Research in Computer Security*, ESORICS '08, pages 333–348, Berlin, Heidelberg, 2008. Springer-Verlag.

[3] T. H. Austin and C. Flanagan. Efficient purely-dynamic information flow analysis. In *Proceedings of the 4th ACM SIGPLAN Workshop on*

Programming Languages and Analysis for Security, PLAS '09, pages 113–124, New York, NY, USA, 2009. ACM.

[4] T. H. Austin and C. Flanagan. Permissive dynamic information flow analysis. In *Proceedings of the 5th ACM SIGPLAN Workshop on Programming Languages and Analysis for Security*, PLAS '10, pages 3:1–3:12, New York, NY, USA, 2010. ACM.

[5] A. Azevedo de Amorim, M. Dénès, N. Giannarakis, C. Hritcu, B. C. Pierce, A. Spector-Zabusky, and A. Tolmach. Micro-policies: Formally verified, tag-based security monitors. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 813–830, May 2015.

[6] A. Bedford, S. Chong, J. Desharnais, E. Kozyri, and N. Tawbi. A progress-sensitive flow-sensitive inlined information-flow control monitor (extended version). *Computers & Security*, 71:114–131, 2017.

[7] A. Bedford, S. Chong, J. Desharnais, and N. Tawbi. A progress-sensitive flow-sensitive inlined information-flow control monitor. In *Proceedings of the 31st IFIP TC 11 International Conference, SEC 2016*, pages 352–366. Springer International Publishing, 2016.

[8] E. D. Bell and J. L. LaPadula. Secure computer systems: Mathematical foundations, 1973.

[9] L. Beringer. End-to-end multilevel hybrid information flow control. In *Proceedings of the 10th Asian Symposium on Programming Languages and Systems*, pages 50–65, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[10] A. Bichhawat, V. Rajani, D. Garg, and C. Hammer. Generalizing permissive-upgrade in dynamic information flow analysis. In *Proceedings of the 9th Workshop on Programming Languages and Analysis for Security*, PLAS '14, pages 15:15–15:24, New York, NY, USA, 2014. ACM.

[11] N. Bielova and T. Rezk. A taxonomy of information flow monitors. In *Proceedings of the 5th International Conference on Principles of Security and Trust, Volume 9635*, pages 46–67, Berlin, Heidelberg, 2016. Springer-Verlag.

[12] P. Buiras, D. Stefan, and A. Russo. On dynamic flow-sensitive floating-label systems. In *Proceedings of the 27th IEEE Computer Security Foundations Symposium*, CSF '14, pages 65–79, Washington, DC, USA, 2014. IEEE Computer Society.

[13] P. Buiras, D. Vytiniotis, and A. Russo. HLIO: Mixing static and dynamic typing for information-flow control in Haskell. In *Proceedings of the 20th ACM SIGPLAN International Conference on Functional Programming*, ICFP 2015, pages 289–301, New York, NY, USA, 2015. ACM.

[14] A. Chudnov and D. A. Naumann. Information flow monitor inlining. In *Proceedings of the 23rd IEEE Computer Security Foundations Symposium*, pages 200–214, July 2010.

[15] A. Chudnov and D. A. Naumann. Inlined information flow monitoring for JavaScript. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, pages 629–643, New York, NY, USA, 2015. ACM.

[16] J. A. Goguen and J. Meseguer. Security policies and security models. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 11–20, 1982.

[17] G. L. Guernic. Precise dynamic verification of confidentiality. In *Proceedings of the 5th International Verification Workshop*, pages 82–96, 2008.

[18] D. Hedin, L. Bello, and A. Sabelfeld. Value-sensitive hybrid information flow control for a JavaScript-like language. In *Proceedings of the 28th IEEE Computer Security Foundations Symposium*, pages 351–365, July 2015.

[19] D. Hedin and A. Sabelfeld. Information-flow security for a core of JavaScript. In *Proceedings of the 25th IEEE Computer Security Foundations Symposium*, CSF '12, pages 3–18, Washington, DC, USA, 2012. IEEE Computer Society.

[20] C. Hritcu, M. Greenberg, B. Karel, B. C. Pierce, and G. Morrisett. All your IFCException are belong to us. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 3–17, May 2013.

[21] S. Hunt and D. Sands. On flow-sensitive security types. In *Conference Record of the 33rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '06, pages 79–90, New York, NY, USA, 2006. ACM.

[22] E. Kozyri, J. Desharnais, and N. Tawbi. Block-safe information flow control. Technical report, Cornell University, 2016.

[23] G. Le Guernic, A. Banerjee, T. Jensen, and D. A. Schmidt. Automata-based confidentiality monitoring. In *Proceedings of the 11th Asian Computing Science Conference on Advances in Computer Science: Secure Software and Related Issues*, ASIAN '06, pages 75–89, Berlin, Heidelberg, 2007. Springer-Verlag.

[24] L. Lourenço and L. Caires. Dependent information flow types. In *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '15, pages 317–328, New York, NY, USA, 2015. ACM.

[25] J. Magazinius, A. Russo, and A. Sabelfeld. On-the-fly inlining of dynamic security monitors. *Computers and Security*, 31(7):827–843, Oct. 2012.

[26] S. Moore and S. Chong. Static analysis for efficient hybrid information-flow control. In *Proceedings of the 24th IEEE Computer Security Foundations Symposium (CSF)*, pages 146–160, 2011.

[27] A. C. Myers. JFlow: Practical mostly-static information flow control. In *Proceedings of the 26th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '99, pages 228–241, New York, NY, USA, 1999. ACM.

[28] A. C. Myers, L. Zheng, S. Zdancewic, S. Chong, and N. Nystrom. Jif 3.0: Java information flow. Software release. http://www.cs.cornell.edu/jif, July 2006.

[29] D. E. Robling Denning. *Cryptography and Data Security*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1982.

[30] A. Russo and A. Sabelfeld. Dynamic vs. static flow-sensitive security analysis. In *Proceedings of the 23rd IEEE Computer Security Foundations Symposium*, CSF '10, pages 186–199, Washington, DC, USA, 2010. IEEE Computer Society.

[31] A. Russo and A. Sabelfeld. Dynamic vs. static flow-sensitive security analysis. In *Proceedings of the 23rd IEEE Computer Security Foundations Symposium*, pages 186–199, July 2010.

[32] A. Sabelfeld and A. Russo. From dynamic to static and back: Riding the roller coaster of information-flow control research. In *Proceedings of the 7th International Andrei Ershov Memorial Conference on Perspectives of Systems Informatics*, PSI '09, pages 352–365, Berlin, Heidelberg, 2010. Springer-Verlag.

[33] J. F. Santos and T. Rezk. An information flow monitor-inlining compiler for securing a core of JavaScript. In *Proceedings of the 29th IFIP TC 11 International Conference, SEC 2014*, pages 278–292, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[34] P. Shroff, S. Smith, and M. Thober. Dynamic dependency monitoring to secure information flow. In *Proceedings of the 20th IEEE Computer Security Foundations Symposium*, CSF '07, pages 203–217, Washington, DC, USA, 2007. IEEE Computer Society.

[35] D. Stefan, D. Mazières, J. C. Mitchell, and A. Russo. Flexible dynamic information flow control in the presence of exceptions. *Journal of Functional Programming*, 27:e5, 2017.

[36] D. Volpano and G. Smith. Eliminating covert flows with minimum typings. In *Proceedings of the 10th Computer Security Foundations Workshop*, pages 156–168, June 1997.

[37] D. M. Volpano and G. Smith. A type-based approach to program security. In *Proceedings of the 7th International Joint Conference CAAP/FASE on Theory and Practice of Software Development*, TAPSOFT '97, pages 607–621, London, UK, 1997. Springer-Verlag.

[38] L. Zheng and A. C. Myers. Dynamic security labels and static information flow control. *International Journal of Information Security*, 6(2):67–84, Mar 2007.