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.10, 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

\[
\text{if } m > 0 \text{ then } w := h \text{ else } w := l \text{ end (1)}
\]

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 \(L \sqsubseteq M \sqsubseteq H\) 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 \(m \neq 0\) 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:

\[
\text{if } m > 0 \text{ then } w := h \text{ else } w := l \text{ end;}
\]
\[
(2)
\]
\[
\begin{array}{c}
\quad m := w; \quad l := 1
\end{array}
\]

---

\(^1\)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.\footnote{In fact, an arbitrary number of bits can be leaked through blocking executions [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 metlabel 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 metlabel (i.e., $M$) would be allowed to observe the label of $w$. The metlabel 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 $\ell_1$, metlabel $\ell_2$, meta-meta label $\ell_3$, 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 $\infty$-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 $\infty$-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...
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 \(k\text{-Enf}\) 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: \(k\text{-precise, } k\text{-varying,}\) and \(k\text{-dependent.}\)

2 Label Chains

Each variable \(x\) in a program will be associated with a possibly infinite label chain \(⟨\ell_1, \ell_2, \ldots, \ell_i, \ell_{i+1}, \ldots⟩\), where label \(\ell_1\) specifies sensitivity for the value stored in \(x\) and label \(\ell_{i+1}\) specifies sensitivity for \(\ell_i\). Labels come from a possibly infinite underlying lattice \(L = ⟨L, \sqsubseteq, \sqcup⟩\) with bottom element \(\bot\).

For any \(\ell, \ell′ \in L\), if \(\ell \sqsubseteq \ell′\) holds, then \(\ell′\) is at least as restrictive as \(\ell\), signifying that information is allowed to flow from data tagged with \(\ell\) to data tagged with \(\ell′\).

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

Unless a label chain \(⟨\ldots, \ell_i, \ell_{i+1}, \ldots⟩\) 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 \(⟨L, H, \ldots⟩\), where \(L \sqsubseteq H\). 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 \(⟨L, H, \ldots⟩\) 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 \(\text{dom}(M)\) 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.

\(^3\text{When } L = ⟨L, \sqsubseteq, \sqcup⟩, \text{ we write } \ell \in L \text{ to assert that } \ell \in L \text{ holds.}\)
by $M$. Let $\text{Var}$ be the set of variables. Constants (e.g., 1, 2, 3) are a subset of $\text{Var}$ 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 $\mu_1, \mu_2, \text{etc.}$

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., $T^i(q) \subseteq \ell, T^i(q) \sqcup T^j(q')$). We require:

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

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

$$M(e \oplus e') \triangleq M(e) \oplus M(e') \quad (3)$$

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

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:

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

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 $\bot$) 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 $\nu$ is a special case of an anchor variable:

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

## 3 Enforcers

Execution of a command $C$ on a memory $M$ can be represented by a trace $\tau$, 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 $\tau'$ of states is considered a subtrace of $\tau$ iff $\tau = \ldots \rightarrow \tau' \rightarrow \ldots$. We write $|\tau|$ to denote the length of $\tau$ and $\tau[i]$ to denote the $i$th state in $\tau$ 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:

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

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 $\nu$. 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. 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.

\footnote{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.}
(Constants) \( \nu \in \mathbb{Z} \)

(Anchor variables) \( a, x \in \text{Var}_A \)

(Flexible variables) \( w, x \in \text{Var}_F \)

(Expressions) \( e ::= \nu \mid x \mid e_1 \oplus e_2 \)

_Commands_

\[ C ::= \text{skip} \mid x := e \mid C_1; C_2 \mid \]
\[ \text{if } e \text{ then } C_1 \text{ else } C_2 \text{ end } \mid \]
\[ \text{while } e \text{ do } C \text{ end } \]

Figure 1: Syntax

\[ (\text{skip}) \quad \langle \text{skip}, M \rangle \rightarrow \langle \text{stop}, M \rangle \]

\[ (\text{Asgn}_A) \quad \nu = M(e) \quad \langle a := e, M \rangle \rightarrow \langle \text{stop}, M[a \mapsto \nu] \rangle \]

\[ (\text{Asgn}_F) \quad \nu = M(e) \quad \langle w := e, M \rangle \rightarrow \langle \text{stop}, M[w \mapsto \nu] \rangle \]

\[ (\text{If}1) \quad M(e) \neq 0 \quad \langle \text{if } e \text{ then } C_1 \text{ else } C_2 \text{ end}, M \rangle \rightarrow \langle C_1; \text{exit}, M \rangle \]

\[ (\text{If}2) \quad M(e) = 0 \quad \langle \text{if } e \text{ then } C_1 \text{ else } C_2 \text{ end}, M \rangle \rightarrow \langle C_2; \text{exit}, M \rangle \]

\[ (\text{Wl}1) \quad M(e) \neq 0 \quad \langle \text{while } e \text{ do } C \text{ end}, M \rangle \rightarrow \langle C; \text{while } e \text{ do } C \text{ end}; \text{exit}, M \rangle \]

\[ (\text{Wl}2) \quad M(e) = 0 \quad \langle \text{while } e \text{ do } C \text{ end}, M \rangle \rightarrow \langle \text{exit}, M \rangle \]

\[ (\text{Exit}) \quad \langle \text{exit}, M \rangle \rightarrow \langle \text{stop}, M \rangle \]

\[ (\text{Seq}1) \quad \langle C_1, M \rangle \rightarrow \langle \text{stop}, M' \rangle \quad \langle C_1; C_2, M \rangle \rightarrow \langle C_2, M' \rangle \]

\[ (\text{Seq}2) \quad \langle C_1, M \rangle \rightarrow \langle C_1', M' \rangle \quad C_1' \neq \text{stop} \quad \langle C_1; C_2, M \rangle \rightarrow \langle C_1'; C_2, M' \rangle \]

Figure 2: Structural operational semantics \( R \)
The rules comprising $R$ define a function $\text{trace}_R(C, M)$ 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 $\text{trace}_R(C, M)$ to be well-defined, $M$ should be healthy for $C$ denoted $M \models H(C)$ and formalized below, where $x \in C$ indicates that $x \in \text{Var}$ appears in $C$:

$$M \models H(C) \triangleq \forall x \in \text{Var}: (x \in C \Rightarrow x \in \text{dom}(M)) \land (x \in \text{dom}(M) \Rightarrow M(x) \in \mathbb{Z}).$$

By definition, if $\text{trace}_R(C, M)$ is finite, then it ends with normal termination state $\langle \text{stop}, M' \rangle$.

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

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

$$\tau \preceq \tau' \triangleq |\tau| \leq |\tau'| \land l = |\tau| \land (\forall 1 \leq i < l: \tau[i] = 0 \tau'[i]) \land (\neg \text{blk}(\tau) \Rightarrow \tau[l] = 0 \tau'[l]).$$

$E$ is an enforcer on $R$ for $P$ if

- $(\forall C, M: \text{trace}_E(C, M) \preceq \text{trace}_R(C, M))$ and
- the image of function $\text{trace}_E$, which is a set of traces, satisfies $P$.

An enforcer $E$ might employ metadata, including label chains of size $n_E \geq 1$ and a set $\text{Aux}_E$ of auxiliaries.

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

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

$$\forall x \in \text{dom}(M) \cap \text{Var}: \\
(\forall 1 \leq i \leq n_E: T^i(x) \in \text{dom}(M) \land M(T^i(x)) \in \mathcal{L}) \land (\forall i > n_E: T^i(x) \notin \text{dom}(M))$$
- \(\text{dom}(M)\) contains requisite auxiliaries \(\text{Aux}E\):

\[
\forall \mu \in \text{Aux}E: \mu \in \text{dom}(M)
\]

- 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 \(x \in \text{dom}(M)\), then sensitivity \(T^{n_E+1}(x)\) of last element \(T^{n_E}(x)\) does not belong to \(\text{dom}(M)\), and thus, \(T^{n_E+1}(x)\) is not defined.

For mapping \(\text{Init}_E\) from auxiliaries in \(\text{Aux}E\) to initial values, a memory \(M\) is defined to be \textit{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 \(\text{Aux}E\) are initialized according to \(\text{Init}_E\):

\[
\forall \mu \in \text{Aux}E: M(\mu) = \text{Init}(\mu).
\]

We consider enforcers \(E \triangleq \langle \text{trace}_E, n_E, \text{Aux}_E, \text{Init}_E \rangle\) satisfying:

\(\textbf{(E1)}\) Trace \(\text{trace}_E(C, M)\) is defined when \(M \models \mathcal{H}_0(E, \mathcal{L}, C)\) holds.

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

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

\(^5\)More formally, if \(\langle C_1; C_2, M_1 \rangle \xrightarrow{\delta} \langle C'_1; C_2, M_2 \rangle\) is a subtrace of \(\text{trace}_E(C, M)\), where \(C_1\) is a subcommand of \(C\), and if "\(w := e\)" \(\not\in C_1\) holds for a flexible variable \(w\), then the following should hold:

\[
(\forall i \geq 1: T^i(w) \in \text{dom}(M_1) \Rightarrow M_1(T^i(w)) = M_2(T^i(w)))
\]

Note, a conditional delimiter (e.g., \textbf{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 $\ell$ observes updates to variables and tags $q$, where $T(q)$ is in the domain of a memory $M$ and $M(T(q)) \subseteq \ell$ holds. A similar threat model is used in [5].

Principals do not observe updates to an identifier $q$ when $T(q) \not\in \text{dom}(M)$ 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 $\ell$ might be allowed to observe updates to an identifier $T^j(q)$ (i.e., $T^{j+1}(q) \not\subseteq \ell$) but $p$ might not be allowed to observe updates to a preceding identifier $T^i(q)$ (i.e., $T^{i+1}(q) \not\subseteq \ell$) for $0 \leq i < j$, due to monotonically decreasing label chains.

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

$$M|_\ell^S \triangleq \{ \langle q, M(q) \rangle \mid q \in S \land T(q) \in \text{dom}(M) \land M(T(q)) \subseteq \ell \}$$

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

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

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

$$\langle C, M \rangle \rightarrow \langle C', M' \rangle \rightarrow \tau \rangle|_\ell^S \triangleq$$

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

When $S = \{ T^i(x) \mid 0 \leq i \leq k \land x \in \text{Var} \}$, we abbreviate $\tau|_\ell^S$ by $\tau|_\ell^k$. We write $\theta =_{\text{abs}} \theta'$ to specify equality of sequences of observations with empty observations omitted, since $\theta$ and $\theta'$ 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 $\ell$, then observations (involving variables and tags) visible to a principal assigned label $\ell$ 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 $M|_{\text{dom}(M)}$.

$$k\text{-BNI}(E, \mathcal{L}, C) \triangleq (\forall \ell \in \mathcal{L} : \forall M, M' :$$

$$M \models \mathcal{H}_0(E, \mathcal{L}, C)$$

$$\land M' \models \mathcal{H}_0(E, \mathcal{L}, C)$$

$$\land M|_{\ell} = M'|_{\ell}$$

$$\land \tau = \text{trace}_E(C, M) \text{ is finite}$$

$$\land \tau' = \text{trace}_E(C, M') \text{ is finite}$$

$$\Rightarrow \tau|_{\ell}^k =_{\text{obs}} \tau'|_{\ell}^k$$

If $k\text{-BNI}(E, \mathcal{L}, C)$ holds for every $C$, then $E$ enforces $k\text{-BNI}(\mathcal{L})$.\footnote{Notice that if $E$ satisfies $(k + 1)\text{-BNI}(\mathcal{L})$, then $E$ satisfies $k\text{-BNI}(\mathcal{L})$.} If for all $k \geq 0$ and $\mathcal{L}$, enforcer $E$ satisfies $k\text{-BNI}(\mathcal{L})$, 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
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 $\infty$-Enf

We use familiar insights about information flow to formulate an enforcer $\infty$-Enf that uses infinite label chains (i.e., $n_{\infty-Enf} = \infty$) to enforce BNI for programs written in the programming language of Figure 1. We later derive from $\infty$-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, $\infty$-Enf updates tag $T(w)$ with $T(e)$. But with that update, the value of $T(e)$ flows explicitly to $T(w)$, so $\infty$-Enf also must update tag $T^2(w)$ with $T^2(e)$. Repeating the argument, we conclude that when executing $w := e$, enforcer $\infty$-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:

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

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 $T(ctx) \subseteq T^{i+1}(w)$ 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 \( ctx \):

\[
UT(q, e, ctx) \triangleq T(q) := T(e) \sqcup T(ctx);
\]

For \( w := e \) in context \( ctx \), \( UT(w, e, ctx) \) expands to

\[
\forall i \geq 1: T_i(w) := T_i(e) \sqcup T(ctx).
\]  \hspace{1cm} (8)

So, enforcer \( \infty\)-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 \( e \): \( T(e) \subseteq T(a) \). 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 \( T(ctx) \sqcup T(G_{a:=e}) \subseteq T(a) \). So, for \( G_{a:=e} \) to hold, both \( T(e) \subseteq T(a) \) and \( T(ctx) \sqcup T(G_{a:=e}) \subseteq T(a) \) should hold. We conclude

\[
G_{a:=e} \Rightarrow (T(e) \subseteq T(a) \land T(ctx) \sqcup T(G_{a:=e}) \subseteq T(a))
\]

---

7This expansion uses the fact that the label chain associated with \( ctx \) is monotonically decreasing: \( T^{i+1}(ctx) \subseteq T^i(ctx) \).
or equivalently

\[ G_{a:=e} \Rightarrow (T(e) \sqcup T(ctx) \sqcup T(G_{a:=e}) \sqsubseteq T(a)). \]  

(9)

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

\[ G_{a:=e} \triangleq (T(e) \sqcup T(ctx) \sqsubseteq T(a)). \]

(10)

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

\[
T(G_{a:=e}) = T(T(e) \sqcup T(ctx) \sqsubseteq T(a)) \\
= T^2(e) \sqcup T^2(ctx) \sqcup T^2(a) \quad \{\text{due to (4)}\} \\
= T^2(e) \sqcup T^2(ctx) \sqcup \bot \\
= T^2(e) \sqcup T^2(ctx) \quad \{T^2(a) = \bot\} \\
= T^2(e) \sqcup T^2(ctx) \quad \{\ell \sqcup \bot = \ell\}
\]

(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 \( \infty\text{-Enf} \)

Enforcer \( \infty\text{-Enf} \) 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. \( \infty\text{-Enf} \) uses auxiliaries to maintain \( T(ctx) \):

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

- \textit{bc} (blocking context) keeps track of the sensitivity of information revealed by block conditions that might influence reachability of the next command executed.
So, $\text{Aux}_{\infty-\text{Enf}} = \{cc, bc\}$. We now show how $T(\text{ctx})$ 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 $\text{ctx}$ in which $G_{a:=e}$ is evaluated. So, whenever a block condition $G_{a:=e}$ is checked, $\infty$-$\text{Enf}$ updates $bc$ with $T(G_{a:=e})$ and $T(\text{ctx})$:

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

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

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

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

Auxiliary $cc$ is implemented in $\infty$-$\text{Enf}$ 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. $|cc|$ 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 $\epsilon$ with $|\epsilon| \triangleq \perp$. So, we have $\text{Init}_{\infty-\text{Enf}}(cc) = \epsilon$.

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

$$T(e) \sqcup |cc| \sqcup bc \sqsubseteq T(a).$$  \hfill (14)

Substituting $|cc| \sqcup bc$ for $T(\text{ctx})$ in (13), the update of $bc$ becomes:

$$bc := T^2(e) \sqcup |cc| \sqcup bc.$$  \hfill (15)

So, $G_{a:=e}$ and the update of $bc$ have now been expressed in terms of tags and auxiliaries that $\infty$-$\text{Enf}$ 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 \textbf{block} versus due to a \textbf{stop}) without sensitive information leaking.
\[
\begin{align*}
\text{(ASGN)} & \quad v = M(e) \quad G_{a:=e} \quad \ell = M(T^2(e)) \sqcup M(\lfloor cc \rfloor) \sqcup M(bc) \\
& \quad \langle a := e, M \rangle \rightarrow \langle \text{stop}, M[a \mapsto v, bc \mapsto \ell] \rangle \\
\text{(ASGNFAIL)} & \quad v = M(e) \quad \neg G_{a:=e} \quad \ell = M(T^2(e)) \sqcup M(\lfloor cc \rfloor) \sqcup M(bc) \\
& \quad \langle a := e, M \rangle \rightarrow \langle \text{block}, M[bc \mapsto \ell] \rangle \\
\text{(ASGNF)} & \quad v_0 = M(e) \quad \forall i \geq 1: v_i = M(T^i(e)) \sqcup M(\lfloor cc \rfloor) \sqcup M(bc) \\
& \quad \langle w := e, M \rangle \rightarrow \langle \text{stop}, M[\forall i \geq 0: T^i(w) \mapsto v_i] \rangle \\
\end{align*}
\]

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

Figure 3: Operational semantics for \textbf{skip} and assignments.

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

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

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 \(M(cc).\text{push}(\langle \ell, W, A \rangle)\)); 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 \(\ell\) is the sensitivity of the guard \(e\) of conditional command \(C\). Including \(\ell\) 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 \(\text{targetFlex}(C_u)\) 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 \(C_t\).
\[
M(e) \neq 0 \quad W = \text{targetFlex}(C_2) \\
A = \text{targetAnchor}(C_2) \quad cc' = M(cc).\text{push}((M(T(e)), W, A))
\]

\[
\begin{array}{l}
\text{(if } e \text{ then } C_1 \text{ else } C_2 \text{ end}, M) \rightarrow (C_1; \text{exit}, M[cc \mapsto cc']) \\
M(e) = 0 \quad W = \text{targetFlex}(C_1) \\
A = \text{targetAnchor}(C_1) \quad cc' = M(cc).\text{push}((M(T(e)), W, A))
\end{array}
\]

\[
\begin{array}{l}
\text{(if } e \text{ then } C_1 \text{ else } C_2 \text{ end}, M) \rightarrow (C_2; \text{exit}, M[cc \mapsto cc']) \\
M(e) \neq 0 \quad cc' = M(cc).\text{push}((M(T(e)), \emptyset, \emptyset))
\end{array}
\]

\[
\begin{array}{l}
\text{(while } e \text{ do } C \text{ end}, M) \rightarrow (\text{while } e \text{ do } C \text{ end; exit}, M[cc \mapsto cc']) \\
M(e) = 0 \quad W = \text{targetFlex}(C) \\
A = \text{targetAnchor}(C) \quad cc' = M(cc).\text{push}((M(T(e)), W, A))
\end{array}
\]

\[
\text{(while } e \text{ do } C \text{ end, } M) \rightarrow (\text{exit}, M[cc \mapsto cc'])
\]

\[
bc' = \begin{cases} 
M(bc) \cup M([cc]), & \text{if } M(cc).\text{top}.A \neq \emptyset \\
M(bc), & \text{otherwise}
\end{cases}
\]

\[
\text{(exit)} \\
\begin{array}{l}
M' = U(M, M(cc).\text{top}.W) \quad cc' = cc.\text{pop} \\
\langle \text{exit}, M \rangle \rightarrow \langle \text{stop}, M'[cc \mapsto cc', bc \mapsto bc'] \rangle
\end{array}
\]

\[
U(M, W) \triangleq \\
M[\forall w \in W: \forall i \geq 1: T^i(w) \rightarrow T^i(w) \cup M([cc] \cup M(bc))]
\]

Figure 4: Operational semantics for conditional commands.

- Element \( A \) is set \( \text{targetAnchor}(C_u) \) 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 \((\text{SeqF})\) asserts that execution stops once an assignment is blocked.

Given a lattice \( \mathcal{L} \), a command \( C \), and a memory \( M \) initially healthy for \( \infty-\text{Enf} \), \( \mathcal{L} \), and \( C \), function \( \text{trace}_{\infty-\text{Enf}}(C, M) \) is defined by the operational...
Figure 5: Operational semantics for sequences

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

**Theorem 1.** $\infty$-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 $\Omega' = \langle \ell_1, \ldots, \ell_{k+1}, \ell_{k+2}, \ldots \rangle$, where $k^{th}$ label $\ell_k$ is infinitely repeated. It is a conservative approximation, because if $\Omega'$ allows a principal $p$ assigned label $\ell$ to observe the $i^{th}$ element of $\Omega'$, then $\Omega$ allows $p$ to observe the $i^{th}$ element of $\Omega$, too (but not *vice versa*). This is because $\Omega$ and $\Omega'$ agree up to the $k^{th}$ element and, for $i \geq k$, the $i^{th}$ element in $\Omega'$ is at least as restrictive as the corresponding element in $\Omega$ due to monotonically decreasing label chains: $\ell_{k+1} \subseteq \ell_k$, $\ell_{k+2} \subseteq \ell_k$, etc. Finite label chain with $m \geq 0$:

$$\Omega' = \langle \ell_1, \ldots, \ell_k, \ell_k, \ldots, \ell_k \rangle_m$$

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

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

\[
\begin{align*}
\text{(SEQ1)} & \quad \langle C_1, M \rangle \rightarrow \langle \text{stop}, M' \rangle \\
& \quad \langle C_1; C_2, M \rangle \rightarrow \langle C_2, M' \rangle \\
\text{(SEQ2)} & \quad \langle C_1, M \rangle \rightarrow \langle C'_1, M' \rangle \quad C' \notin \{\text{stop, block}\} \\
& \quad \langle C_1; C_2, M \rangle \rightarrow \langle C'_1; C_2, M' \rangle \\
\text{(SEQF)} & \quad \langle C_1, M \rangle \rightarrow \langle \text{block}, M' \rangle \\
& \quad \langle C_1; C_2, M \rangle \rightarrow \langle \text{block}, M' \rangle
\end{align*}
\]
\[ v_0 = M(e) \quad \forall 1 \leq i \leq k: v_i = M(T^i(e)) \sqcup M([cc]) \sqcup M(bc) \]

\[ (w := e, M) \rightarrow (\text{stop}, M[\forall i: 0 \leq i \leq k: T^i(w) \mapsto v_i]) \]

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

Figure 6: Modified rules for \( k \)-Enf

compute up to the \( k^{th} \) tag. Because rule \( \text{(ASGNF)} \) mentions \( T^2(x) \), we require \( k \geq 2 \). In \( \infty \)-Enf, only \( \text{(ASGNF)} \) and function \( U \) refers to \( T^i(x) \) for \( i > 2 \). So in \( k \)-Enf rule \( \text{(ASGNF)} \) 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 \( \infty \)-Enf with finite label chain \( \Omega'' = \langle \ell_1, \ldots, \ell_k, \ell_k \rangle \).

Similar to \( \infty \)-Enf, enforcer \( k \)-Enf has \( \text{Aux}_{k,\text{Enf}} = \{ cc, bc \} \), \( \text{Init}_{k,\text{Enf}}(cc) = \epsilon \), and \( \text{Init}_{k,\text{Enf}}(bc) = \perp \). 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 \( \theta \) of observations produced by
$E$, also appear in a sequence $\theta'$ produced by $E'$. We define $\theta \preceq \theta'$ as follows

\[
\theta \preceq \theta' \triangleq \quad |\theta| \leq |\theta'| \land (\forall i: 1 \leq i \leq |\theta|: \text{dom}(\theta[i]) \subseteq \text{dom}(\theta'[i]))
\]

where $\theta[i]$ is the $i$th observation in sequence $\theta$. Relation $\preceq$ 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 $\rho$, underlying lattice $L$, and identifiers up to the $k$th tag (i.e., $T^k$) with $k \geq 0$, writing $E \leq_{\rho}^{k,L} E'$, as follows:

\[
E \leq_{\rho}^{k,L} E' \triangleq \forall C, M, M': \rho(M, M')
\land M \models \mathcal{H}_0(E, L, C) \land M' \models \mathcal{H}_0(E', L, C)
\Rightarrow (\forall \ell \in \mathcal{L}: \text{trace}_E(C, M)|_{\ell}^{k} \subseteq \text{trace}_{E'}(C, M')|_{\ell}^{k})
\]

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

For convenience, we introduce abbreviations:

- $E \prec_{\rho}^{k,L} E' \triangleq E \leq_{\rho}^{k,L} E' \land E' \not\leq_{\rho}^{k,L} E$
- $E \sim_{\rho}^{k,L} E' \triangleq E \leq_{\rho}^{k,L} E' \land E' \leq_{\rho}^{k,L} E$

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

\[
E \leq_{\rho'}^{k,L} E' \Rightarrow E \leq_{\rho}^{k,L} E'.
\]

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

We now examine how lengths of label chains relate to the permissiveness of enforcers by comparing the permissiveness of enforcers $k$-$\text{Enf}$ and $(k+1)$-$\text{Enf}$ 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 $k$th. Define

$$M^{|k} ≜ \{⟨T^i(x), M(T^i(x))⟩ | 0 \leq i \leq k \land x \in \text{Var} \land T^i(x) \in \text{dom}(M)\}$$

The desired initialization condition then is:

$$\rho_k(M, M') ≜ M^{|k} = M'^{|k}$$

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

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

We say that $\Omega$ exhibits a $(k + 1)$-decrease because $T^{k+1}(w) \sqsubseteq T^k(w)$. 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, $\Omega'$ is a conservative approximation of $\Omega$.

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

**Theorem 3.** $k$-$Enf <_{\rho_k}^{k, \mathcal{L}} (k + 1)$-$Enf$, 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 $k$-$Enf <_{\rho_k}^{k, \mathcal{L}} \infty$-$Enf$, 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., $⟨\bot, \ldots, \bot⟩$). We say memory $M$ is conventionally initialized when for set $\text{Var}_F$ of flexible variables

$$\alpha_c(M) ≜ \forall w \in \text{Var}_F: \forall i \geq 1: \quad T^i(w) \in \text{dom}(M) \Rightarrow M(T^i(w)) = \bot.$$
We also define initialization condition
\[ c(M, M') \triangleq \alpha_c(M) \land \rho_1(M, M') \]
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 \( <^k_L \rho_k \) is replaced with \( <^k_c \rho_k \). With initialization condition \( c \), label chains longer than two elements do not enhance the permissiveness of \( k\text{-Enf} \). This is because, for initialization condition \( c \), enforcer \( k\text{-Enf} \) produces label chains where the second element is always repeated\(^8\) (e.g., \( \langle H, M, M, \ldots, M \rangle \)) due to the conservative update of label chains of flexible variable induced by rules \((\text{AsgnF})\) in Figure 3 and \((\text{Exit})\) 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.** \( k\text{-Enf} \simeq_{^c_L}^k (k + 1)\text{-Enf} \) for any lattice \( L \) 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 \( k\text{-Enf} \) satisfies \( k\text{-BNI} \). So, \( k\text{-Enf} \) satisfies 0-BNI, because 0-BNI is implied by \( k\text{-BNI} \).

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

**Theorem 5.** \( k\text{-Enf} \simeq_{^0_L}^c (k + 1)\text{-Enf} \) for any lattice \( L \) and \( k \geq 2 \).

Because \( c \Rightarrow \rho_k \) holds, property (17) and Theorem 5 gives \( k\text{-Enf} \simeq_{^0_L} (k + 1)\text{-Enf} \) for any lattice \( L \) and \( k \geq 2 \). So, under the weakened threat model and for both initialization conditions (i.e, \( \rho_k \) and \( c \)), the permissiveness of \( k\text{-Enf} \) 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

<table>
<thead>
<tr>
<th>Threat Model</th>
<th>ρ_k</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strong</td>
<td>✓</td>
<td>x</td>
</tr>
<tr>
<td>Weak</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

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 \( ⟨\ell_1, \ell_2, \ldots, \ell_k⟩ \) and \( ⟨\ell'_1, \ell'_2, \ldots, \ell'_k⟩ \) with labels from lattice \( L \), are defined to be k-varying for \( k \geq 2 \) iff

\[
(\forall i: 1 \leq i < k: \ell_i = \ell'_i) \land \ell_k \neq \ell'_k.
\]

Notice that k-varying label chains cannot exist when \( L \) has only one element.

We now formalize k-precise. Consider an enforcer \( E \), lattice \( L \), command \( C \), and conventionally initialized memory \( M \) such that \( M \models \mathcal{H}_0(E, L, C) \).
Assume trace $\tau = \text{trace}_E(C, M)$ produces label chain prefix $\Omega = \langle \ell_1, \ldots, \ell_n \rangle$ at some state $\tau[j]$ after an assignment to a flexible variable $w$:

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

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

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

then

- trace $\tau' = \text{trace}_{E'}(C, M')$ 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 $\Omega$ is $k$-precise, then any enforcer $E'$ that satisfies $(k - 1)$-BNI($\mathcal{L}$) and is at least as permissive as $E$ (i.e., $E \preceq_{c}^{k-1,\mathcal{L}} E'$) will produce (at the same execution point) the same first $k$ elements that appear in $\Omega$. Consequently, the first $k$ elements of $\Omega$ 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 $\Omega, \Omega'$ such that:

- $\Omega$ is $k$-precise at the $i$th state of $\text{trace}_E(C, M)$, for some $i$ and $M$ with $M \models \mathcal{H}_0(E, \mathcal{L}, C)$,
- $\Omega'$ is $k$-precise at the $j$th state of $\text{trace}_E(C', M')$, for some $j$ and $M'$ with $M' \models \mathcal{H}_0(E, \mathcal{L}, C')$,
- $\Omega$ and $\Omega'$ 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 $k$th 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 $f_{E'}$:

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

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,

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

(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 $k$-$E_{opt}$, 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 $k$-$E_{opt}$ 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 $T^i(w) = \perp$ 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 $k$-$E_{opt}$ 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 $(a := e; C', M')$ in a trace $\text{trace}_E(C, M)$. Boolean expression $G^{E}_{a:=e}$ is satisfied in a memory $M'$ according to

\[
M'(G^{E}_{a:=e}) \iff \langle a := e; C', M' \rangle \rightarrow \langle C', M'' \rangle \text{ is a subtrace of } \text{trace}_E(C, M).
\]

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 $a := e$. 

26
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 \( V_{a=e} \):

\[
G^E_{a=e} = f_E(\{T^i(x) \mid x \in V_{a=e} \land 1 \leq i \leq k\}),
\]

for some function \( f_E \). For example, \( 2\text{-Enf} \) 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 \( 2\text{-Enf} \). 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 \( 2\text{-Enf} \).

**Theorem 7.** For an enforcer \( E \) and lattice \( \mathcal{L}_3 \equiv (\{L, M, H\}, \sqsubseteq, \sqcup) \), if \( G^E_{a=e} \) is 1-dependent and \( 2\text{-Enf} \preceq_0^{c,\mathcal{L}_3} E \), 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 \( 2\text{-Enf} \), 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., \( E_{H,L} \) in the next section) that use 1-dependent \( G^E_{a=e} \), are at least as permissive as \( 2\text{-Enf} \) 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 \( \mathcal{L}_2 \equiv (\{L, H\}, \sqsubseteq, \sqcup) \) with \( L \subseteq H \), 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 $L_2$ 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 $E_{HL}$, which is derived from $k$-$Enf$ by associating each variable with only one tag. Figure 8 shows the modified rules for $E_{HL}$. We prove below (Theorem 8) that $G_{a:=e}$ defined in Figure 8 is 1-dependent.

$E_{HL}$ 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 $M(\lfloor cc \rfloor) = H$. So, $T(w)$ will reveal no information about the value of that sensitive guard. Thus, the sensitivity of $T(w)$ is $L$.

Define function $trace_{E_{HL}}(C, M)$ to map command $C$ and memory $M$ with $M \models H_0(E_{HL}, L, C)$ to the entire trace that starts with state $\langle C, M \rangle$. We have $n_{E_{HL}} = 1$, $Aux_{E_{HL}} = \{cc, bc\}$, $Init_{E_{HL}}(cc) = e$, and $Init_{E_{HL}}(bc) = \bot$.

Theorem 7 does not hold when $L_3$ is replaced with $L_2$ if $E$ is $E_{HL}$. Instead, Theorem 8 below holds; it states that $E_{HL}$ satisfies 0-BNI and is strictly more permissive than 2-$Enf$ only when $L_2$ is used.

**Theorem 8.** Enforcer $E_{HL}$ uses 1-dependent $G_{a:=e}$, satisfies 0-BNI($L_2$), and satisfies 2-$Enf <_{0,L_2} E_{HL}$.

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 $L_2$ does not necessarily generalize for arbitrary lattices.

Notice, though, that $E_{HL}$ does not satisfy 0-BNI for arbitrary lattices. For example, consider (2), which employs $L_3$. Based on rules in Figure 8 and rules (iv),(vii) in §5.3, $E_{HL}$ executes (2) as described in (i) and (ii) in §1. So, executing (2) under $E_{HL}$ leaks sensitive $m > 0$ to principals observing non-sensitive variable $l$, and thus, 0-BNI is not satisfied. $E_{HL}$ 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.
\begin{align*}
\text{(AsgnA)} & \quad v = M(e) \quad G_{a:=e} \quad \ell = M([cc]) \sqcup M(bc) \\
& \quad (a := e, M) \rightarrow \langle \text{stop}, M[a \mapsto v, bc \mapsto \ell] \rangle \\
\text{(AsgnAFail)} & \quad v = M(e) \quad \neg G_{a:=e} \quad \ell = M([cc]) \sqcup M(bc) \\
& \quad (a := e, M) \rightarrow \langle \text{block}, M[bc \mapsto \ell] \rangle \\
\text{(AsgnF)} & \quad v_0 = M(e) \quad v_1 = M(T(e)) \sqcup M([cc]) \sqcup M(bc) \\
& \quad (w := e, M) \rightarrow \langle \text{stop}, M[w \mapsto v_0, T(w) \mapsto v_1] \rangle \\
U(M, W) & \triangleq M[\forall w \in W: T(w) \mapsto T(w) \sqcup M([cc]) \sqcup M(bc)] \\
G_{a:=e} & \text{ is } M(T(e)) \sqcup M([cc]) \sqcup M(bc) \sqsubseteq M(T(a))
\end{align*}

Figure 8: Modified rules for \( E_{H,L} \)

<table>
<thead>
<tr>
<th>Threat Model</th>
<th>Label Chain Length</th>
<th>Greater than 1</th>
<th>Greater than 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strong</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Weak (( L_3 ))</td>
<td>✓</td>
<td>✓</td>
<td>?</td>
</tr>
<tr>
<td>Weak (( L_2 ))</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>

Figure 9: ✓ indicates where labels chains with length greater than the one indicated in the corresponding column can provide enhanced permissiveness; x 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 \( k\)-\textit{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 \( L_2 \) 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 \( 2\text{-Enf} \). We are also not aware of an enforcement mechanism that uses label chains of length one, enforces \( L_2 \), satisfies BNI, and is at least as permissive as \( E_{H,L} \).

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 \texttt{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., \( 3\text{-Eopt} \)).

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 $L_2$, modify or skip the execution of an unsafe command. Similar to [12], the enforcement mechanism presented in [25] (which enforces labels from $L_2$) 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 $2$-$\textit{Enf}$, 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 $\leq^k_{\rho,L}$ is more general than the one presented in [31], because it is defined on any two enforcers and handles arbitrary lattices (not just $L_2$) 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 $L_2$. 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 $L_2$.

A Definitions

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

- We abbreviate $\tau|_S$ 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 $\ell$ to include $bc$ (blocking context) and $cc$ (conditional context):

$$M|_{\ell} =$$

$$\{\langle q, M(q) \rangle \mid q, T(q) \in \text{dom}(M) \land M(T(q)) \subseteq \ell \} \cup$$

$$\{\langle bc, M(bc) \rangle \mid M(bc) \subseteq \ell \} \cup$$

$$\{\langle cc, M(cc) \rangle \mid M([cc]) \sqcup M(bc) \subseteq \ell \}$$

(18)
• Define equality of sequences of observations when omitting the empty sets:

\[ \epsilon =_{\text{obs}} \epsilon \]
\[ \emptyset =_{\text{obs}} \epsilon \]
\[ \epsilon =_{\text{obs}} \emptyset \]

\[ \Theta \rightarrow \theta =_{\text{obs}} \Theta' \rightarrow \theta' \text{ iff } \]
\[ \begin{cases} 
\Theta = \Theta' \land \theta =_{\text{obs}} \theta', \text{ or } \\
\Theta = \emptyset \land \Theta' \neq \emptyset \land \theta =_{\text{obs}} \Theta' \rightarrow \theta', \text{ or } \\
\Theta \neq \emptyset \land \Theta' = \emptyset \land \Theta \rightarrow \theta =_{\text{obs}} \theta'
\end{cases} \]

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

\[ kstut(M) \triangleq \forall x \in \text{dom}(M): \forall i > k: T^i(x) \in \text{dom}(M) \Rightarrow M(T^i(x)) = M(T^k(x)) \]

for \( k \geq 1 \)

• We write \( \text{mon}(M) \) to denote that all label chains in \( M \) are monotonically decreasing:

\[ \text{mon}(M) \triangleq \forall x \in \text{dom}(M): \forall i \geq 1: T^{i+1}(x) \in \text{dom}(M) \Rightarrow M(T^{i+1}(x)) \subseteq M(T^i(x)) \]

• We write \( M =_k M' \) iff

1. \( \forall x: \forall 0 \leq i \leq k: M(T^i(x)) = M'(T^i(x)) \),
2. \( M(\text{cc}) = M'(\text{cc}) \), and
3. \( M(\text{bc}) = M'(\text{bc}) \).

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

\text{BNI+}(E, L, C). \forall \ell \in L: \forall M, M':

\text{If}

\[ M \models \mathcal{H}(E, L, C) \land M' \models \mathcal{H}(E, L, C), \]
\[ M|_\ell = M'|_\ell, M(\text{cc}) = M'(\text{cc}), \text{mon}(M), \text{mon}(M') \]
\[ \tau = \text{trace}_E(C, M), \]
\[ \tau' = \text{trace}_E(C, M'), \]

33
where \( \tau = \langle C, M \rangle \xrightarrow{\ast} \langle C_t, M_t \rangle, \) \( \tau' = \langle C, M' \rangle \xrightarrow{\ast} \langle C'_t, M'_t \rangle, \) and \( C_t, C'_t \) are terminations (i.e., stop or block), then:

- c1 If \( C_t \) and \( C'_t \) are both stop, then \( \tau|_\ell =_{\text{obs}} \tau'|_\ell, \) \( M_t|_\ell = M'_t|_\ell, \) and \( M_t(cc) = M'_t(cc). \)
- c2 If \( C_t \) or \( C'_t \) is block, then \( \tau|_\ell =_{\text{obs}} \tau'|_\ell. \)
- c3 If \( C_t \) is stop, \( C'_t \) is block, and \( M'_t(bc) \not\subseteq \ell, \) then \( M_t(bc) \not\subseteq \ell. \)
- c4 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 \( \tau', \) and \( M'_{tp}(cc) \subseteq \ell, \) then there exists \( \langle C'', M'' \rangle \in \tau, \) with \( C'' = C''_{tp} \) and \( M''|_\ell = M'_{tp}|_\ell. \)

- For enforcers \( k\text{-Enf} \) with \( 2 \leq k \leq \infty, \) \( E_{H,L}, \) and \( k\text{-Eopt} \) with \( k \geq 2, \) we extend the domain of function \( \text{trace}_E(C, M) \) to also include any memory \( M \) such that \( M \models \mathcal{H}(E, L, C) \) holds.

Also, if \( M \models \mathcal{H}(E, L, C) \) holds, then for any \( M' \) in \( \text{trace}_E(C, M) \) and any subcommand \( C' \) of \( C, \) we have \( M' \models \mathcal{H}(E, L, C') \) (based on the corresponding operational semantics).

## B Soundness of \( \infty\text{-Enf} \) and \( k\text{-Enf} \) for \( k \geq 2 \)

**Theorem 1.** \( \infty\text{-Enf} \) is an enforcer on \( R \) for BNI.

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

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

**Lemma 1.** For a command \( C \) and lattice \( L, \) \( k\text{-BNI+}(\infty\text{-Enf}, L, C) \) holds.

**Proof.** Let \( \ell \in L. \) We use structural induction on \( C. \)
1. \( C \) is \texttt{skip}:
   From rule \texttt{skip}, we get \( C_t = C'_t = \texttt{stop} \). So, \( c2, c3, \) and \( c4 \) are trivially true.

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

2. \( C \) is \( a := e \):
   From \( M|_t = M'|_t \) and \( M(T^2(a)) = M'(T^2(a)) = \bot \), we get
   \[
   M(T(a)) = M'(T(a)).
   \]
   (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 \texttt{AsgnA} in Figure 3 has been triggered, meaning that \( M(T(e)) \sqcup M([cc]) \sqcup M(bc) \sqsubseteq M(T(a)) \) holds. From hypothesis (2.1.), we then get \( M(T(e)) \sqsubseteq \ell \), \( M([cc]) \sqsubseteq \ell \), and \( M(bc) \sqsubseteq \ell \).

   Because \( M|_t = M'|_t \), we then get \( M(cc) = M'(cc) \) and \( M(bc) = M'(bc) \). From \( \text{mon}(M) \) 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 \( M|_t = M'|_t \), 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 \( M'(T(e)) \sqcup M'( [cc]) \sqcup M'(bc) \sqsubseteq M'(T(a)) \) holds. So, rule \texttt{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:
   \[
   \tau = \langle a := e, M \rangle \rightarrow \langle \texttt{stop}, M[a \mapsto M(e), bc \mapsto \ell_g] \rangle \\
   \tau' = \langle a := e, M' \rangle \rightarrow \langle \texttt{stop}, M'[a \mapsto M'(e), bc \mapsto \ell_g'] \rangle,
   \]

35
where
\[
\ell_g = M(T^2(e)) \cup M([cc]) \sqcup M(bc) \quad \text{and} \\
\ell_g' = M'(T^2(e)) \cup M'([cc]) \sqcup M'(bc).
\]

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 \(M_t(cc) = M'(cc) = M'_t(cc)\), 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.
\[
\tau = \langle a := e, M \rangle \rightarrow \langle \text{block}, M[bc \mapsto \ell_g] \rangle
\]
\[
\tau' = \langle a := e, M' \rangle \rightarrow \langle \text{block}, M'[bc \mapsto \ell'_g] \rangle.
\]
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 \(\tau|_\ell = \text{obs} \epsilon\) and \(\tau'|_\ell = \text{obs} \epsilon\), and thus c2 holds.
To prove c1, assume \(C_t = C'_t = \text{stop}\). 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 \(M_k(bc) \not\sqsubseteq \ell\), then \(M_t|_\ell = M'_t|_\ell\) trivially holds. Assume instead \(M_k(bc) \sqsubseteq \ell\). So, rule \textsc{AsgnA} in Figure 3 then gives \(M(T^2(e)) \cup M([cc]) \sqcup M(bc) \sqsubseteq \ell\).
Thus, \(M(T^2(e)) \sqsubseteq \ell, M([cc]) \sqsubseteq \ell, M(bc) \sqsubseteq \ell\). From \textit{mon(M)} and \(M(T^2(e)) \sqsubseteq \ell\), we get \(M(T^3(e)) \sqsubseteq \ell\). From \(M_t|_\ell = M'_t|_\ell\), \(M(T^3(e)) \sqsubseteq \ell\), and Lemma 5, we get \(M(T^2(e)) = M'(T^2(e))\).
From \(M_t|_\ell = M'_t|_\ell, M([cc]) \sqsubseteq \ell, M(bc) \sqsubseteq \ell\), we get: \(M([cc]) = M'([cc])\) and \(M(bc) = M'(bc)\). Because \textsc{AsgnA} in Figure 3 gives \(M_k(bc) = M(T^2(e)) \sqcup M([cc]) \sqcup M(bc)\) and \(M'_k(bc) = M'(T^2(e)) \sqcup M'([cc]) \sqcup M'(bc)\), we then have \(M_k(bc) = M'_k(bc)\). From \(M_k(cc) = M'_k(cc)\), we then get \(M_k|_\ell = M'_k|_\ell\). So, c1 holds.

To prove c3, assume \(C_t\) is \text{stop}, \(C'_t\) is \text{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 \text{stop}, \(C'_t\) is \text{block}, \(\langle C'_{tp}, M'_{tp} \rangle \rightarrow \langle C_t, M'_t \rangle\) are the last two states of \(\tau'\), and \(M'_{tp}([cc]) \sqsubseteq \ell\). So, \(M'_{tp} = M'\) and
\[ C_t' = a := e. \] We have that \((C'', M'')\) in c4 is \((a := e, M)\), which satisfies \(C'' = C_t'\) and \(M''|_\ell = M'|_\ell\). Thus c4 holds.

3. \(C\) is \(w := e\)
\[
\tau = \langle w := e, M \rangle \rightarrow \langle \text{stop}, M_t \rangle \\
\tau' = \langle w := e, M' \rangle \rightarrow \langle \text{stop}, M'_t \rangle
\]
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. \(\exists r \geq 1: M_t(T^r(w)) \subseteq \ell\)
From \(\text{mon}(M)\), we then get \(\forall i \geq r: M_t(T^i(w)) \subseteq \ell\). Then we have \(\forall i \geq r: M_t(T^i(e)) \subseteq \ell\), \(M_t(cc) \subseteq \ell\), and \(M_t(bc) \subseteq \ell\). From \(M_t|_\ell = M'_t|_\ell\) and Lemma 5, we then get \(\forall i \geq r - 1: M_t(T^i(e)) = M'_t(T^i(e)), M_t(cc) = M'_t(cc)\), and \(M_t(bc) = M'_t(bc)\). So, \(\forall i \geq r - 1: M_t(T^i(w)) = M'_t(T^i(w))\), and thus, \(\forall i \geq r: M'_t(T^i(w)) \subseteq \ell\).
- 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 \(\forall i: 1 < i < r: M_t(T^i(w)) \not\subseteq \ell\). Because \(\forall i \geq r - 1: M_t(T^i(w)) = M'_t(T^i(w))\), we then get \(M'_t(T^{i-1}(w)) \not\subseteq \ell\). From \(\text{mon}(M')\) we then get \(\forall i < r: M'_t(T^i(w)) \not\subseteq \ell\). Thus, \(M_t|_\ell = M'_t|_\ell\) and \(\tau|_\ell = \tau'|_\ell\). Thus c1 holds.

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

4. \(C_1; C_2\)

4.1. \(C_1\) terminates normally in \(\tau\) and \(\tau'\).
Let
\[
\tau = \langle C_1; C_2, M \rangle \rightarrow \langle C_2, M_2 \rangle \rightarrow \langle C_{2t}, M_t \rangle \quad \text{and} \\
\tau' = \langle C_1; C_2, M' \rangle \rightarrow \langle C_2, M'_2 \rangle \rightarrow \langle C'_{2t}, M'_t \rangle.
\]
Consider:
\[
\tau_1 = \langle C_1, M \rangle \rightarrow \langle \text{stop}, M_2 \rangle, \\
\tau_2 = \langle C_2, M_2 \rangle \rightarrow \langle C_{2t}, M_t \rangle, \\
\tau'_1 = \langle C_1, M' \rangle \rightarrow \langle \text{stop}, M'_2 \rangle, \\
\tau'_2 = \langle C_2, M'_2 \rangle \rightarrow \langle C'_{2t}, M'_t \rangle.
\]
From $C_1$ of IH on $C_1$, we get $\tau_1|_\ell = \text{obs } \tau_1'|_\ell$, $M_2|_\ell = M_2'|_\ell$, and $M_2((cc) = M_2'(cc)$. From $\text{mon}(M)$, $\text{mon}(M')$ and Lemma 2, we get $\text{mon}(M_2)$, $\text{mon}(M_2')$. So, we can apply IH on $C_2$.

To prove $c_1$, assume $C_{2t} = C_{2t}' = \text{stop}$. From IH on $C_2$, we get $\tau_2|_\ell = \text{obs } \tau_2'|_\ell$, $M_t|_\ell = M_t'|_\ell$, and $M_t((cc) = M_t'(cc)$. From $\tau_1|_\ell = \text{obs } \tau_1'|_\ell$ and $\tau_2|_\ell = \text{obs } \tau_2'|_\ell$, we get $\tau|_\ell = \text{obs } \tau'|_\ell$. Thus $c_1$ holds.

To prove $c_2$, assume $C_{2t} = \text{block}$ or $C_{2t}' = \text{block}$. From IH on $C_2$, we get $\tau_2|_\ell = \text{obs } \tau_2'|_\ell$. From $\tau_1|_\ell = \text{obs } \tau_1'|_\ell$ and $\tau_2|_\ell = \text{obs } \tau_2'|_\ell$, we get $\tau|_\ell = \text{obs } \tau'|_\ell$. Thus $c_2$ holds.

To prove $c_3$, assume $C_{2t} = \text{stop}$, $C_{2t}' = \text{block}$, and $M_t(bc) \not\subseteq \ell$. From IH on $C_2$, we get $M_t(bc) \not\subseteq \ell$. Thus $c_3$ holds.

To prove $c_4$, assume $C_{2t} = \text{stop}$, $C_{2t}' = \text{block}$, $\langle C_{tp}', M_{tp}' \rangle \to C_{2t}', M_t'$ are the last two states of $\tau'$, and $M_{tp}'(cc) \notsubseteq \ell$. Then $\langle C_{tp}', M_{tp}' \rangle \to C_{2t}', M_t'$ are the last two states of $\tau_2'$. 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 $c_4$ holds.

4.2. $C_1$ is blocked in both $\tau, \tau'$

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

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

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:

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

$\tau_1' = \langle C_1, M' \rangle \xrightarrow{*} \langle \text{stop}, M_t' \rangle$,

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

So, we have

$\tau|_\ell = \tau_1|_\ell$ and $\tau'|_\ell = \tau_1'|_\ell \rightarrow \tau_2'|_\ell$. (20)

We prove $c_3$. We have $C_t = C_{1t} = \text{block}$. Assume $C_{2t}' = \text{stop}$ and $M_t(bc) \not\subseteq \ell$. We prove $M_t'(bc) \not\subseteq \ell$. For IH on $C_1$, we get $M_2'(bc) \not\subseteq \ell$. From Lemma 7, we then get $M_t'(bc) \not\subseteq \ell$. Thus $c_3$ holds.

We prove $c_4$. We have $C_t = \text{block}$. Assume $C_{2t}' = \text{stop}$, that $\langle C_{tp}; C_2, M_{tp} \rangle \to \langle C_{1t}; C_2, M_t \rangle$ are the last two states of $\tau$, and that
5. \textbf{if} \textit{e} \textbf{then} \textit{C}_1 \textbf{else} \textit{C}_2 \textbf{end}

5.1. \( M([cc]) \subseteq \ell \) and \( M(T(e)) \subseteq \ell \)

From \textit{mon}(\( M \)) and \( M(T(e)) \subseteq \ell \), we have \( M(T^2(e)) \subseteq \ell \). Because \( M|_\ell = M'|_\ell \) and Lemma 5, we then have \( M(T(e)) = M'(T(e)) \) and \( M(e) = M'(e) \). So, \( \tau \) and \( \tau' \) get the same branch, say \( \textit{C}_1 \).

\( \tau = \langle \textbf{if} \textit{e} \textbf{then} \textit{C}_1 \textbf{else} \textit{C}_2 \textbf{end}, M \rangle \rightarrow \langle \textit{C}_1; \textit{exit}, M_1 \rangle \rightarrow \langle \textit{C}_1, M_t \rangle \)

\( \tau' = \langle \textbf{if} \textit{e} \textbf{then} \textit{C}_1 \textbf{else} \textit{C}_2 \textbf{end}, M' \rangle \rightarrow \langle \textit{C}_1; \textit{exit}, M'_1 \rangle \rightarrow \langle \textit{C}_1', M'_t \rangle \).

From \( M_1([cc]) = M([cc]).\text{push}(\langle M(T(e)), W, A \rangle) \), \( M([cc]) = M'(cc) \), \( M(T(e)) = M'(T(e)) \), and \( M'_1([cc]) = M'(cc).\text{push}(\langle M'(T(e)), W, A \rangle) \), 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 $M_1(bc) \nsubseteq \ell$. Assume $M_1(bc) \subseteq \ell$. 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 $M(bc) \nsubseteq \ell$. 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 $\text{mon}(M)$, $\text{mon}(M')$ and Lemma 2, we get $\text{mon}(M_1)$, $\text{mon}(M'_1)$. We apply IH on $C_1: \text{exit}$ and get $c_1$, $c_2$, $c_3$, and $c_4$.

5.2. $M([cc]) \nsubseteq \ell$ or $M(T(\epsilon)) \nsubseteq \ell$

We first prove that $M'(\{cc\}) \nsubseteq \ell$ or $M'(T(\epsilon)) \nsubseteq \ell$ holds. If $M(T(\epsilon)) \nsubseteq \ell$, then from $M|_{\ell} = M'_1$ and Lemma 6, we get $M'(T(\epsilon)) \nsubseteq \ell$. Now, if $M([cc]) \nsubseteq \ell$, then $M(cc) = M'(cc)$ gives $M'(\{cc\}) \nsubseteq \ell$. Thus, we have $M'(\{cc\}) \nsubseteq \ell$ or $M'(T(\epsilon)) \nsubseteq \ell$.

So, Lemma 3 gives:

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

(ii) If $C_1 = \text{stop}$ (or $C'_1 = \text{stop}$), and $w \in \text{targetFlex}(C)$, then $\forall i: M_t(T^i(w)) \nsubseteq \ell$ (or $\forall i: M'_t(T^i(w)) \nsubseteq \ell$).

(iii) If $C_1 = \text{stop}$ (or $C'_1 = \text{stop}$), and $\text{targetAnchor}(C) \neq \emptyset$ then $M_t(bc) \nsubseteq \ell$ or $(M'_t(bc) \nsubseteq \ell)$.

So, $c_2$ holds.

We prove $c_1$. Assume $C_1 = C'_1 = \text{stop}$. Because $\tau|_{\ell} = \text{obs} \epsilon$ and $\tau'|_{\ell} = \text{obs} \epsilon$, it suffices to prove $M_t|_{\ell} = M'_t|_{\ell}$ and $M_t(cc) = M'_t(cc)$.

From Lemma 8, we get $M(cc) = M'_1(cc)$ and $M'(cc) = M'_1(cc)$.

From $M'(cc) = M'(cc)$, we then get $M_t(cc) = M'_t(cc)$. We prove $M_t|_{\ell} = M'_t|_{\ell}$. If $M_t(T^{i+1}(x)) \subseteq \ell$, then (ii) gives $x \notin \text{targetFlex}(C)$.

So, $M_t(T^i(x)) = M'(T^i(x))$, $M'_t(T^i(x)) = M'_t(T^{i+1}(x))$, $M_t(T^{i+1}(x)) = M(T^{i+1}(x))$, $M'_t(T^{i+1}(x)) = M'_t(T^{i+1}(x))$. Thus, $M(T^{i+1}(x)) \subseteq \ell$.

From $M|_{\ell} = M'_1|_{\ell}$, we then have $M'(T^{i+1}(x)) \subseteq \ell$ and $M(T^{i+1}(x)) = M'(T^{i+1}(x))$. By transitivity, $M_t(T^i(x)) = M'_t(T^i(x))$ and $M'_t(T^{i+1}(x)) \subseteq \ell$.

Assume $M_t(bc) \subseteq \ell$. So, (iii) gives $\text{targetAnchor}(C) = \emptyset$. So, $M(bc) = M_t(bc)$ and $M'(bc) = M'_t(bc)$. From $M_t(bc) \subseteq \ell$, we then get $M(bc) \subseteq \ell$. From $M|_{\ell} = M'_1|_{\ell}$, we then get $M(bc) = M'(bc)$. Thus, $M_t(bc) = M'_t(bc)$. So, $M_t|_{\ell} = M'_t|_{\ell}$. Thus $c_1$ holds.

We prove $c_3$. If $C'_1 = \text{block}$, then $\text{targetAnchor}(C) \neq \emptyset$. So, (iii) gives $M_t(bc) \not\subseteq \ell$. Thus $c_3$ holds.
We prove $c_4$. We have that $M'(\langle cc \rangle) \not\sqsubseteq \ell$ or $M'(T(e)) \not\sqsubseteq \ell$. So, if $C' = \textbf{block}$, then $M'_\text{tp}(\langle cc \rangle) \not\sqsubseteq \ell$. Thus $c_4$ holds.

6. while $e$ do $C_1$ end

Induction on the maximum number of iterations in $\tau$ and $\tau'$.

Base case: Both $\tau$ and $\tau'$ take (Wl2).

$\tau = \langle \textbf{while } e \textbf{ do } C_1 \textbf{ end, } M \rangle \rightarrow \langle \textbf{exit, } M_e \rangle \rightarrow \langle \textbf{stop, } M_t \rangle$

$\tau' = \langle \textbf{while } e \textbf{ do } C_1 \textbf{ end, } M' \rangle \rightarrow \langle \textbf{exit, } M'_e \rangle \rightarrow \langle \textbf{stop, } M'_t \rangle$.

So, $c_2, c_3, c_4$ are trivially true.

We prove $c_1$. We have $\tau|_t = \tau'(t_e)$ and $\tau'|_t = \tau'(t_e)$.

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 $M|_t = M'|_t$ and Lemma 5, we then get $M(T(e)) = M'(T(e))$. Both $\tau$ and $\tau'$ 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 $M_e|_t = M'_e|_t$. If $M_e(bc) \not\sqsubseteq \ell$, then it is trivial, given $M|_t = M'|_t$. 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 $M|_t = M'|_t$, 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 $M_e|_t = M'_e|_t$. 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 $M_e|_t = M'_e|_t$ and $M_e(cc) = M'_e(cc)$. Thus $c_1$ 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 $M'(T(e)) \not\sqsubseteq \ell$. 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 $M_t|_t = M'_t|_t$. Using Lemma 3, if $M_t(T^{i+1}(x)) \sqsubseteq \ell$, then $x \not\in \text{targetFlex}(C)$. So, $M_t(T^i(x)) = M(T^i(x))$, $M'_t(T^i(x)) = M'_t(T^i(x))$, 41
\( M_i(T^{i+1}(x)) = M(T^{i+1}(x)) \), \( M'(T^{i+1}(x)) = M'_i(T^{i+1}(x)) \). Thus, \( M(T^{i+1}(x)) \subseteq \ell \). From \( M|_\ell = M'|_\ell \), we then have \( M'(T^{i+1}(x)) \subseteq \ell \) and \( M(T^i(x)) = M'(T^i(x)) \). By transitivity, \( M_i(T^i(x)) = M'_i(T^i(x)) \) and \( M'_i(T^{i+1}(x)) \subseteq \ell \). Assume \( M_i(bc) \subseteq \ell \). From Lemma 3, we then get that \( \text{targetAnchor}(C) = \emptyset \). So, \( M(bc) = M_i(bc) \) and \( M'(bc) = M'_i(bc) \). From \( M_i(bc) \subseteq \ell \) and Lemma 7, we get \( M(bc) \subseteq \ell \). From \( M|_\ell = M'|_\ell \), we then get \( M(bc) = M'(bc) \). Thus, by transitivity, we get \( M_i(bc) = M'_i(bc) \). And because \( M_i(cc) = M'_i(cc) \), we consequently have \( M_i|_\ell = M'_i|_\ell \). Thus c1 holds.

**Induction case:**

6.1. \( M([cc]) \subseteq \ell \) and \( M(T(e)) \subseteq \ell \)

From \( \text{mon}(M) \), we then have \( M(T^2(e)) \subseteq \ell \). From \( M|_\ell = M'|_\ell \) and Lemma 5, we then get \( M(T(e)) = M'(T(e)) \) and \( M(e) = M'(e) \). So, \( \tau \) and \( \tau' \) take the same branch. If both take (w.l.o.g.), then we follow the Base case.

Assume that both take (w.l.o.g.):

\[ \tau = \langle \text{while } e \text{ do } C_1 \text{ end}, M \rangle \rightarrow \langle C_1; \text{ while } e \text{ do } C_1 \text{ end}; \text{ exit}, M_1 \rangle \rightarrow \langle C_t, M_t \rangle, \]

\[ \tau' = \langle \text{while } e \text{ do } C_1 \text{ end}, M' \rangle \rightarrow \langle C_1; \text{ while } e \text{ do } C_1 \text{ end}; \text{ exit}, M'_1 \rangle \rightarrow \langle C'_t, M'_t \rangle. \]

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 \( M_1(bc) \subseteq \ell \). Because \( M_1(bc) = M(bc) \), we then get \( M(bc) \subseteq \ell \). 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 \( \text{mon}(M_i) \) and \( \text{mon}(M'_i) \), from Lemma 2, \( \text{mon}(M) \), and \( \text{mon}(M') \).

6.1.1. \( C_1 \) terminates normally in the 1st iteration in \( \tau \) and \( \tau' \).

\[ \tau = \langle \text{while } e \text{ do } C_1 \text{ end}, M \rangle \rightarrow \langle C_1; \text{ while } e \text{ do } C_1 \text{ end}; \text{ exit}, M_1 \rangle \rightarrow \langle \text{while } e \text{ do } C_1 \text{ end}; \text{ exit}, M_2 \rangle \rightarrow \langle C_t, M_t \rangle, \]

\[ \tau' = \langle \text{while } e \text{ do } C_1 \text{ end}, M' \rangle \rightarrow \langle C_1; \text{ while } e \text{ do } C_1 \text{ end}; \text{ exit}, M'_1 \rangle \rightarrow \langle \text{while } e \text{ do } C_1 \text{ end}; \text{ exit}, M'_2 \rangle \rightarrow \langle C'_t, M'_t \rangle. \]

Consider:

\[ \tau_1 = \langle C_1, M_1 \rangle \rightarrow \langle \text{stop}, M_2 \rangle \]

\[ \tau'_1 = \langle C_1, M'_1 \rangle \rightarrow \langle \text{stop}, M'_2 \rangle \]

42
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 $\tau_1|_\ell = t_{\text{obs}} \tau'_1|_\ell, M_2|_\ell = M'_2|_\ell$, 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:

$\tau_2 = \langle \text{while } e \text{ do } C_1 \text{ end, } M_2 \rangle \rightarrow (C_3, M_3)$
$\tau'_2 = \langle \text{while } e \text{ do } C_1 \text{ end, } M'_2 \rangle \rightarrow (C'_3, M'_3)$

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 $\tau_2$ and $\tau'_2$.

We prove $c2$. Say that $C_t$ is block. Then $C_3$ should be block. From $IH[c2]$ on $\tau_2$ and $\tau'_2$, we then get $\tau_2|_\ell = t_{\text{obs}} \tau'_2|_\ell$. Because we have $\tau|_\ell = t_{\text{obs}} \tau_1|_\ell \rightarrow \tau_2|_\ell, \tau'|_\ell = t_{\text{obs}} \tau'_1|_\ell \rightarrow \tau'_2|_\ell, \tau_1|_\ell = t_{\text{obs}} \tau'_1|_\ell, \text{ and } \tau_2|_\ell = t_{\text{obs}} \tau'_2|_\ell$, we get $\tau|_\ell = t_{\text{obs}} \tau'|_\ell$. So, $c2$ holds.

We similarly prove $c3$ and $c4$.

We prove $c1$. Assume $\tau$ and $\tau'$ terminate normally:

$\tau = \langle \text{while } e \text{ do } C_1 \text{ end, } M \rangle \rightarrow (C_1; \text{while } e \text{ do } C_1 \text{ end; exit, } M_1)$
$\rightarrow \langle \text{while } e \text{ do } C_1 \text{ end; exit, } M_2 \rangle \rightarrow (\text{exit, } M_3) \rightarrow (\text{stop, } M_4)$
$\tau' = \langle \text{while } e \text{ do } C_1 \text{ end, } M' \rangle \rightarrow (C_1; \text{while } e \text{ do } C_1 \text{ end; exit, } M'_1)$
$\rightarrow (\text{while } e \text{ do } C_1 \text{ end; exit, } M'_2) \rightarrow (\text{exit, } M'_3) \rightarrow (\text{stop, } M'_4)$

Then $\tau_2$ and $\tau'_2$ terminate normally. So, we have $\tau_2 = \langle \text{while } e \text{ do } C_1 \text{ end, } M_2 \rangle \rightarrow (\text{stop, } M_3)$
$\tau'_2 = \langle \text{while } e \text{ do } C_1 \text{ end, } M'_2 \rangle \rightarrow (\text{stop, } M'_3)$

By IH[c1] on $\tau_2$ and $\tau'_2$, we then get $\tau_2|_\ell = t_{\text{obs}} \tau'_2|_\ell, M_3|_\ell = M'_3|_\ell$ and $M_3((cc) = M'_3((cc)$. Because $\tau|_\ell = t_{\text{obs}} \tau_1|_\ell \rightarrow \tau_2|_\ell, \tau'|_\ell = t_{\text{obs}} \tau'_1|_\ell \rightarrow \tau'_2|_\ell, \tau_1|_\ell = t_{\text{obs}} \tau'_1|_\ell, \text{ and } \tau_2|_\ell = t_{\text{obs}} \tau'_2|_\ell$, we get $\tau|_\ell = t_{\text{obs}} \tau'|_\ell$.

To prove $c1$, we also need to prove that $M_4|_\ell = M'_4|_\ell$ and $M_4((cc) = M'_4((cc)$. Consider:

$\tau_3 = (\text{exit, } M_3) \rightarrow (\text{stop, } M_4)$
$\tau'_3 = (\text{exit, } M'_3) \rightarrow (\text{stop, } M'_4)$

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 $M_4|_\ell = M'_4|_\ell$ and $M_4((cc) = M'_4((cc)$. So, $c1$ holds.

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

6.1.3. $C_1$ blocked in $\tau$, terminates normally in $\tau'$. 

43
\[\tau = \langle \text{while } e \text{ do } C_1 \text{ end}, M \rangle \rightarrow \langle C_1; \text{while } e \text{ do } C_1 \text{ end}; \text{exit}, M_1 \rangle \]
\[\rightarrow \langle C_{1t}; \text{while } e \text{ do } C_1 \text{ end}; \text{exit}, M_1 \rangle,\]
\[\tau_1 = \langle \text{while } e \text{ do } C_1 \text{ end}, M' \rangle \rightarrow \langle C_1; \text{while } e \text{ do } C_1 \text{ end}; \text{exit}, M'_1 \rangle \]
\[\rightarrow \langle \text{while } e \text{ do } C_1 \text{ end}; \text{exit}, M'_1 \rangle \rightarrow \langle C'_{1t}, M'_1 \rangle.\]

Consider:
\[\tau_1 = \langle C_1, M_1 \rangle \rightarrow \langle C_{1t}, M_t \rangle\]
\[\tau'_1 = \langle C_1, M'_1 \rangle \rightarrow \langle \text{stop}, M'_2 \rangle\]
\[\tau'_2 = \langle \text{while } e \text{ do } C_1 \text{ end}; \text{exit}, M'_2 \rangle \rightarrow \langle C'_{1t}, M'_1 \rangle\]

IH can be applied to \(\tau_1\) and \(\tau'_1\), because \(M_1|_t = M'_1|_t, M_1(cc) = M'_1(cc), mon(M'_1),\) and \(C_1\) is a subcommand of \(\text{while } e \text{ do } C_1 \text{ end}.\)

We invoke (4.3.) for \(\tau_1, \tau'_1,\) and \(\tau'_2.\)

### 6.2. \(M([cc]) \not\subseteq \ell \text{ or } M(T(e)) \not\subseteq \ell\)

Consider:
\[\tau_w = \langle \text{if } e \text{ then } (C_1; \text{while } e \text{ do } C_1 \text{ end}) \text{ else } \text{skip } \text{ end}, M \rangle \]
\[\rightarrow \langle C_wt, M_wt \rangle,\text{ and}\]
\[\tau'_w = \langle \text{if } e \text{ then } (C_1; \text{while } e \text{ do } C_1 \text{ end}) \text{ else } \text{skip } \text{ end}, M'_w \rangle \]
\[\rightarrow \langle C'_w, M'_w \rangle.\]

We invoke case 5.2. for \(\tau_w, \tau'_w,\) and we get:

**c1w** If \(C_wt\) and \(C'_w\) are both \textbf{stop}, then \(\tau_w|_t = obs \tau'_w|_t, M_wt|_t = M'_w|_t,\) and \(M_wt(cc) = M'_w(cc).\)

**c2w** If \(C_wt\) or \(C'_w\) is \textbf{block}, then \(\tau_w|_t = obs \tau'_w|_t.\)

**c3w** If \(C_wt\) is \textbf{stop}, \(C'_w\) is \textbf{block}, and \(M'_w(bc) \not\subseteq \ell,\) then \(M_wt(bc) \not\subseteq \ell.\)

We prove c1. Assume \(C_t\) and \(C'_t\) are both \textbf{stop}. Because \(C_t = C_wt\) and \(C'_t = C'_w,\) we have that \(C_wt\) and \(C'_w\) are both \textbf{stop}. From c1w, we have \(\tau_w|_t = \tau'_w|_t, M_wt|_t = M'_w|_t,\) and \(M_wt(cc) = M'_w(cc).\) We have \(M_t = M_wt, M'_t = M'_w, \tau|_t = obs \tau_w|_t,\) and \(\tau'|_t = obs \tau'_w|_t.\) So, \(\tau|_t = obs \tau'|_t, M_t|_t = M'_t|_t,\) and \(M_t(cc) = M'_t(cc).\) Thus c1 holds.

Similarly, we get c2 and c3.

**c4** is trivially true: from \(M([cc]) \not\subseteq \ell \text{ or } M(T(e)) \not\subseteq \ell\) we get \(M'_w([cc]) \not\subseteq \ell.\)

### 7. exit

We have:
\[ \tau = \langle \text{exit}, M \rangle \rightarrow \langle \text{stop}, M_t \rangle \]
\[ \tau' = \langle \text{exit}, M' \rangle \rightarrow \langle \text{stop}, M'_t \rangle. \]
c2, c3, c4 are trivially true.

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

7.1. \( M_t([cc]) \cup M_t(bc) \not\subseteq \ell \) and \( M(cc).\text{top}.A \neq \emptyset \).

We first prove that \( M_t(bc) \not\subseteq \ell \) and \( M'_t(bc) \not\subseteq \ell \). Because \( M(cc).\text{top}.A \neq \emptyset \), we have \( M_t(bc) = M(bc) \cup M([cc]) \). Because \( M_t([cc]) \subseteq M([cc]) \), we get \( M_t([cc]) \cup M(bc) \subseteq M([cc]) \cup M(bc) \), which becomes \( M_t([cc]) \cup M(bc) \subseteq M([cc]) \cup M(bc) \), which becomes \( M_t([cc]) \cup M_t(bc) \subseteq M([cc]) \cup M(bc) \), due to \( M_t(bc) = M(bc) \cup M([cc]) \).

From \( M_t([cc]) \cup M_t(bc) \not\subseteq \ell \) we get \( M([cc]) \cup M(bc) \not\subseteq \ell \). From \( M_t(bc) = M(bc) \cup M([cc]) \), we then have \( M_t(bc) \not\subseteq \ell \). From \( M([cc]) \cup M(bc) \not\subseteq \ell \) and \( M|_\ell = M'_t|_\ell \), we get \( M'(bc) \cup M(bc) \not\subseteq \ell \). Because \( M(cc).\text{top}.A \neq \emptyset \) and \( M(cc) = M'(cc) \), we have \( M'(bc).\text{top}.A \neq \emptyset \), too. So, \( M'_t(bc) = M'(bc) \cup M'(cc) \). From \( M'(cc) \cup M'(bc) \not\subseteq \ell \), we then have \( M'_t(bc) \not\subseteq \ell \). So, \( M_t(bc) \not\subseteq \ell \) and \( M'_t(bc) \not\subseteq \ell \).

Only variables in \( W \) change their labels. Let \( x \in M(cc).\text{top}.W \). Because \( M([cc]) \cup M(bc) \not\subseteq \ell \), we have \( \forall i \geq 1. M_t(T^i(x)) \not\subseteq \ell \). Because \( M(cc) = M'(cc) \), we get \( x \in M'(cc).\text{top}.W \), too. From \( M'_t(bc) \not\subseteq \ell \), we have \( M'(cc) \cup M'(bc) \not\subseteq \ell \), and thus, \( \forall i \geq 1. M'_t(T^i(x)) \not\subseteq \ell \). So, \( M|_\ell = M'_t|_\ell \). Thus c1 holds.

7.2. \( M_t([cc]) \cup M_t(bc) \not\subseteq \ell \) and \( M(cc).\text{top}.A = \emptyset \).

Because \( M(cc).\text{top}.A = \emptyset \), we have \( M_t(bc) = M(bc) \). We have \( M_t([cc]) \subseteq M([cc]) \). We get \( M_t([cc]) \cup M(bc) \subseteq M([cc]) \cup M(bc) \), which becomes \( M_t([cc]) \cup M_t(bc) \subseteq M([cc]) \cup M(bc) \), due to \( M_t(bc) = M(bc) \). From \( M_t([cc]) \cup M_t(bc) \not\subseteq \ell \), we then have \( M([cc]) \cup M(bc) \not\subseteq \ell \). Because \( M|_\ell = M'_t|_\ell \), we also get \( M'(cc) \cup M'(bc) \not\subseteq \ell \).

We prove that if \( M_t(bc) \subseteq \ell \), then \( M_t(bc) = M'_t(bc) \). Assume \( M_t(bc) \subseteq \ell \). From \( M_t(bc) = M(bc) \), we then get \( M(bc) \subseteq \ell \). From \( M|_\ell = M'_t|_\ell \), we then get \( M(bc) = M'_t(bc) \). Because \( M(cc).\text{top}.A = \emptyset \) and \( M(cc) = M'(cc) \), we have that \( M'(cc).\text{top}.A = \emptyset \). 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 \( M_t([cc]) \cup M_t(bc) \not\subseteq \ell \),

45
we then get $M'_t([cc]) \cup M'_t(bc) \not\subseteq \ell$.

Only variables in $W$ change their labels. Let $x \in M(cc).top.W$. Because $M([cc]) \cup M(bc) \not\subseteq \ell$, we have $\forall i \geq 1. M'_t(T^i(x)) \not\subseteq \ell$. Because $M(cc) = M'(cc)$, we get $x \in M'(cc).top.W$, too. Because $M'(([cc]) \cup M'(bc) \not\subseteq \ell$, we have $\forall i \geq 1. M'_t(T^i(x)) \not\subseteq \ell$. So, $M'_t|_{\ell} = M'_t|_{\ell}$. Thus c1 holds.

7.3. $M'_t([cc]) \cup M'_t(bc) \subseteq \ell$

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

- Let $M([cc]) \cup M(bc) \subseteq \ell$. Let $x \in M(cc).top.W$. Because $M(cc) = M'(cc)$, we get $x \in M'(cc).top.W$. We have: $M_t(T^i(x)) \subseteq \ell \Rightarrow M(T^i(x)) \subseteq \ell \Rightarrow M(T^{i-1}(x)) = M'(T^{i-1}(x))$ and $M'(T^i(x)) \subseteq \ell$.
- 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 $M'_t(T^i(x)) \subseteq \ell$. Thus, $M'_t|_{\ell} = M'_t|_{\ell}$. Thus c1 holds.

- Let $M([cc]) \cup M(bc) \not\subseteq \ell$.
- So, $M'_t([cc]) \cup M'_t(bc) \not\subseteq \ell$. Let $x \in M(cc).top.W$. Consequently, we have that $\forall i \geq 1: M'_t(T^i(x)) \not\subseteq \ell$. Because $M(cc) = M'(cc)$, we get $x \in M'(cc).top.W$, and thus $\forall i \geq 1: M'_t(T^i(x)) \not\subseteq \ell$. Thus, $M'_t|_{\ell} = M'_t|_{\ell}$. Thus c1 holds.

\[ \square \]

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

**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 $\langle C, M \rangle \xrightarrow{\ast} \langle C', M' \rangle$. We prove the statement for one-step transition, using induction on the rules of $\infty$-$Enf$’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, $mon(M')$.

\[ 46 \]
3. \text{(AsgnAFail)}:
   Same arguments as in above case.

4. \text{(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
   \[ \forall i \geq 1: M(T^{i+1}(e)) \cup M([cc]) \sqcup M(bc) \sqsubseteq M(T^i(e)) \cup M([cc]) \sqcup M(bc), \]
   and thus $\forall i \geq 1: M'(T^{i+1}(w)) \sqsubseteq M'(T^i(w))$. So, $mon(M').$

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

6. \text{(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, $M(T^{i+1}(w)) \cup M([cc]) \sqcup M(bc) \sqsubseteq M(T^i(w)) \cup M([cc]) \sqcup M(bc).$ So, $M'(T^{i+1}(w)) \sqsubseteq M'(T^i(w))$. So, $mon(M').$

7. \text{(Seq1),(Seq2),(SeqF)}:
   We use the IH.

\[ \square \]

**Lemma 3.** If $C$ does not include \textbf{exit} (or \textbf{i-exit}), if $\tau = \langle C, M \rangle \rightarrow \langle C', M' \rangle$ is generated by $\infty$-$\text{Enf}$ and $M([cc]) \sqcup M(bc) \not\subseteq \ell,$ or $C$ is a conditional command (executed under $w$, $w'$, or $wl$ rule—not $ws$ rules) with guard $e$ and $M(T(e)) \not\subseteq \ell$, then

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

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

(iii) if $C' = \text{stop}$ and $\text{targetAnchor}(C) \neq \emptyset$, then $M'(bc) \not\subseteq \ell$.

\textbf{Proof.} Induction on $C$ (which should not include \textbf{exit}).

1. $a := e$
   Assume $M([cc]) \sqcup M(bc) \not\subseteq \ell$.
   If $C' = \text{block}$, then $\tau|_\ell = \epsilon$, so (i) holds, and (ii), (iii) are trivially true.
Assume \( C' = \text{stop} \). So, \( M([cc]) \cup M(bc) \subseteq M(T(a)) \). Because \( M([cc]) \cup M(bc) \not\subseteq \ell \), we then get \( M(T(a)) \not\subseteq \ell \). Thus \( \tau|_\ell =_{\text{obs}} \epsilon \) and (i) holds.

(ii) is trivially true.

We have \( M([cc]) \cup M(bc) \subseteq M'(bc) \). Because \( M([cc]) \cup M(bc) \not\subseteq \ell \), we then get \( M'(bc) \not\subseteq \ell \). Thus (iii) holds.

2. \( w := e \)
   Assume \( M([cc]) \cup M(bc) \not\subseteq \ell \).

(iii) is trivially true.

Because \( M([cc]) \cup M(bc) \not\subseteq \ell \) and \( \forall i \geq 1: M([cc]) \cup M(bc) \subseteq M'(T^i(w)) \), we get \( \forall i \geq 1: M'(T^i(w)) \not\subseteq \ell \). Thus (ii) holds.

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

3. \( C_1; C_2 \)
   Assume \( M([cc]) \cup M(bc) \not\subseteq \ell \).

- Assume \( \tau \) involves only the execution of \( C_1 \).
  So, \( \tau \) is blocked, and thus, (ii), (iii) are trivially true.
  We prove (i). Because \( \tau \) involves only the blocked execution of \( C_1 \), consider \( \tau_1 = \langle C_1, M \rangle \xrightarrow{*} \langle \text{block}, M' \rangle \). From IH on \( C_1 \), we get \( \tau_1|_\ell =_{\text{obs}} \epsilon \). Because \( \tau|_\ell = \tau_1|_\ell \), we then get \( \tau|_\ell =_{\text{obs}} \epsilon \). Thus (i) holds.

- Assume \( \tau \) involves execution of \( C_1 \) and \( C_2 \).
  Then \( C_1 \) is executed to normal termination. Consider:
  \( \tau_1 = \langle C_1, M \rangle \xrightarrow{*} \langle \text{stop}, M_1 \rangle \).
  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
  \( M(bc) \subseteq M_1(bc) \). So, \( M(bc) \cup M(cc) \subseteq M_1(bc) \cup M_1([cc]) \). From \( M([cc]) \cup M(bc) \not\subseteq \ell \), we then have \( M_1(bc) \cup M_1([cc]) \not\subseteq \ell \).
  We prove (i). From IH on \( C_1 \), we get \( \tau_1|_\ell =_{\text{obs}} \epsilon \). From IH on \( C_2 \), we get \( \tau_2|_\ell =_{\text{obs}} \epsilon \). Because \( \tau|_\ell = \tau_1|_\ell \rightarrow \tau_2|_\ell \), we get \( \tau|_\ell =_{\text{obs}} \epsilon \). Thus (i) holds.

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

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

4. if \( e \) then \( C_1 \) else \( C_2 \) end

Assume \( M([cc]) \cup M(bc) \not\subseteq \ell \) or \( M(T(e)) \not\subseteq \ell \). W.l.o.g. assume that \( \tau \) involves the execution of \( C_1 \).

So:
\[ \tau = \langle \text{if } e \text{ then } C_1 \text{ else } C_2 \text{ end}, M \rangle \rightarrow \langle C'_1; \text{exit}, M_1 \rangle \rightarrow \langle C'; M' \rangle. \]

Consider:
\[ \tau_1 = \langle C_1, M_1 \rangle \rightarrow \langle C'_1, M'_1 \rangle. \]

We have \( M(bc) = M_1(bc) \) and \( M([cc]) \subseteq M_1([cc]) \). So, \( M([cc]) \cup M(bc) \subseteq M_1([cc]) \cup M_1(bc) \). If \( M([cc]) \cup M(bc) \not\subseteq \ell \), we then have \( M_1(bc) \cup M_1([cc]) \not\subseteq \ell \). If \( M(T(e)) \not\subseteq \ell \), we then have \( M_1(bc) \cup M_1([cc]) \not\subseteq \ell \), because \( M(T(e)) \subseteq M_1([cc]) \). So, in any case, \( M_1(bc) \cup M_1([cc]) \not\subseteq \ell \). So, we can apply IH on \( C_1 \).

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

We prove (ii). Assume \( C' = \text{stop} \) and \( w \in \text{targetFlex}(C) \). Then \( C'_1 = \text{stop} \) and \( w \in \text{targetFlex}(C_1) \) or \( w \in \text{targetFlex}(C_2) \). We have:
\[ \tau = \langle \text{if } e \text{ then } C_1 \text{ else } C_2 \text{ end}, M \rangle \rightarrow \langle C'_1; \text{exit}, M_1 \rangle \rightarrow \langle \text{exit}, M'_1 \rangle \rightarrow \langle C', M' \rangle \quad \text{and} \]
\[ \tau_1 = \langle C_1, M_1 \rangle \rightarrow \langle \text{stop}, M'_1 \rangle. \]

Assume \( w \in \text{targetFlex}(C_1) \). From IH on \( C_1 \), we get \( \forall i \geq 1: M'_1(T^i(w)) \not\subseteq \ell \). Due to the rule for \( \text{exit} \), we get \( \forall i \geq 1: M'_1(T^i(w)) \subseteq M'(T^i(w)) \). So, \( \forall i \geq 1: M'(T^i(w)) \not\subseteq \ell \). Thus (ii) holds.

Assume \( w \not\in \text{targetFlex}(C_1) \). From \( w \in \text{targetFlex}(C) \), we then have \( w \in \text{targetFlex}(C_2) \). So, \( w \in M_1([cc]).top.W \). From Lemma 8, we then have \( w \in M'_1([cc]).top.W \). Due to the rule for \( \text{exit} \), we get \( \forall i \geq 1: M'_1([cc]) \cup
Proof. By structural induction on $\ell$

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

**Proof.** By structural induction on $C$ and Lemma 7. \hfill \Box

**Lemma 5.** If $M|_\ell = M'|_\ell$ and $M(T^i(e)) \subseteq \ell$, 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 $M(T^i(x)) \subseteq \ell$, we have $M(T^{i-1}(x)) = M'(T^{i-1}(x))$.

3. $e$ is $e_1 \oplus e_2$:
   We have $M(T^i(e)) = M(T^i(e_1)) \cup M(T^i(e_2))$. From $M(T^i(e)) \subseteq \ell$, we then get $M(T^i(e_1)) \subseteq \ell$ and $M(T^i(e_2)) \subseteq \ell$. 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 \( M(T^i(e)) = M(T^i(e_1)) \sqcup M(T^i(e_2)) = M'(T^i(e_1)) \sqcup M'(T^i(e_2)) = M'(T^i(e)) \).

\[ \Box \]

**Lemma 6.** If \( M|_\ell = M'|_\ell \), \( \text{mon}(M) \), \( \text{mon}(M') \) 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 \( \text{mon}(M) \), 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. \[ \Box \]

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

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

\[ \Box \]

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

**Proof.** We use structural induction on \( C \). \[ \Box \]

**Lemma 9.** Let \( \langle C, M \rangle \rightarrow \langle C', M' \rangle \) be a trace generated by \( \infty\text{-Enf} \). If \( \ast\text{stut}(M) \), then \( \ast\text{stut}(M') \).

**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 \( \langle C, M \rangle \rightarrow \langle C', M' \rangle \). We prove the statement for one-step transition, using induction on the rules of \( \infty\text{-Enf} \)’s operational semantics. Assume, for all \( x \), we have \( M(T^2(x)) \sqsubseteq M(T(x)) \) and \( \forall i > 1: M(T^i(x)) = M'(T^i(x)) \).

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

2. \((\text{AsgnA})\):
   - 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. \textit{(AsgnAFail)}:
   Same arguments as in above case.

4. \textit{(AsgnF)}:
   Because for every \( x \) we have \( M(T^2(x)) \subseteq M(T(x)) \), we get \( M(T^2(e)) \subseteq M(T(e)) \). We then have
   \( M(T^2(e)) \sqcup M([cc]) \sqcup M(bc) \subseteq M(T(e)) \sqcup M([cc]) \sqcup M(bc) \), and thus \( M'(T^2(w)) \subseteq M'(T(w)) \).
   Similarly, we get \( \forall i > 1: M'(T^2(w)) = M'(T^i(w)) \).

5. \textit{(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. \textit{(Exit)}:
   Only label chains of \( w \in W \) change. We have \( M'(T^2(w)) = M(T^2(w)) \sqcup M([cc]) \sqcup M(bc) \) and \( M'(T(w)) = M(T(w)) \sqcup M([cc]) \sqcup M(bc) \). Because \( M(T^2(w)) \subseteq M(T(w)) \), we then get \( M'(T^2(w)) \subseteq M'(T(w)) \). Similarly, we get \( \forall i > 1: M'(T^2(w)) = M'(T^i(w)) \).

7. \textit{(Seq1),(Seq2),(SeqF)}:
   We use IH.

\[ \square \]

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

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

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

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

We prove \( M|_\ell = M'_I|_\ell \). Due to \( M|_\ell = M'|_\ell, M_I =_k M, \) and \( M'_I =_k M' \), it suffices to examine \( \forall x: \forall i > k: T^i(x) \). Assume \( M_I(T^i(x)) \subseteq \ell \). From \( k\text{stut}(M_I) \), we then get \( M_I(T^k(x)) \subseteq \ell \). From \( M_I =_k M \), we then have \( M(T^k(x)) \subseteq \ell \). By definition of \( k\text{-}\text{Enf} \), we have \( T^{k+1}(x) = T^k(x) \), and thus \( M(T^{k+1}(x)) \subseteq \ell \). From \( M_I =_k M \), we then get \( M(T^k(x)) = M'(T^k(x)) \). From \( M_I =_k M \), \( k\text{stut}(M_I), M'_I =_k M' \), and \( k\text{stut}(M'_I) \), we have \( M_I(T_i^{i-1}(x)) = M'_I(T_i^{i-1}(x)) \) 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 \( \text{mon}(M_I) \), due to \( \text{mon}(M), M_I =_k M \), and \( k\text{stut}(M_I) \). Similarly, we have \( \text{mon}(M'_I) \).

Consider

\[
\tau_I = \text{trace}_{\infty\text{-}\text{Enf}}(C, M_I) = \langle C, M_I \rangle \xrightarrow{*} \langle C, M_I \rangle,
\]

and

\[
\tau'_I = \text{trace}_{\infty\text{-}\text{Enf}}(C, M'_I) = \langle C, M'_I \rangle \xrightarrow{*} \langle C, M'_I \rangle.
\]

From Lemma 10, and applying induction on the number of steps in \( \tau \) and \( \tau' \), we get \( \tau|_k = \tau|_k \) and \( \tau'|_k = \tau'|_k \). Because \( \infty\text{-}\text{Enf} \) satisfies BNI+ (Lemma 1), we get \( \tau|_k = \text{obs } \tau'|_k \). We then get \( \tau|_k = \text{obs } \tau'|_k \). So, \( \tau|_k = \text{obs } \tau'|_k \). \( \square \)

**Lemma 10.** Consider \( \tau = \langle C, M \rangle \rightarrow \langle C_n, M_n \rangle \) be a step under \( k\text{-}\text{Enf} \) with \( k \geq 2 \). Let \( M' =_k M \), \( k\text{stut}(M') \), and \( \tau' = \langle C, M' \rangle \rightarrow \langle C'_n, M'_n \rangle \) be a step under \( \infty\text{-}\text{Enf} \). Then, \( M'_n =_k M_n, k\text{stut}(M'_n) \), \( C_n = C'_n \), and \( \tau|_k = \tau'|_k \).

**Proof.** Structural induction on \( C \).

1. \( C \) is \textbf{skip}:

   We have \( M = M_n \) and \( M' = M'_n \). So, \( M'_n =_k M_n \) and \( k\text{stut}(M'_n) \). Also, \( C_n = C'_n = \textbf{stop} \) and \( \tau|_k = \tau'|_k = \epsilon \).

2. \( C \) is \textbf{a := e}:

   \( G_{a := e} \) is \( T(e) \cup M(|cc|) \cup bc \subseteq T(a) \). 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, \( \tau \) and \( \tau' \) both are generated from \( \text{ASGNA} \), or both from \( \text{ASGNAF ail} \). So, \( C_n = C'_n \). Also, \( M_n(bc) = M'_n(bc) \). Thus, \( M'_n =_k M_n \). Because no label chain changed, \( k\text{stut}(M') \) gives \( k\text{stut}(M'_n) \). Because \( M(e) = M'(e) \), we also get \( \tau|_k = \tau'|_k \).

3. \( C \) is \textbf{w := e}:

   We have \( C_n = C'_n = \textbf{stop} \). We have \( \forall 0 \leq i \leq k: \)
Lemma 11. If \( \langle C, M \rangle \rightarrow \langle C', M' \rangle \) according to \( k\text{-}\text{Enf} \), and \( 2\text{stut}(M) \), then \( 2\text{stut}(M') \).

Proof. Assume \( \langle C, M_I \rangle \rightarrow \langle C'_I, M'_I \rangle \) according to \( \infty\text{-}\text{Enf} \), where \( M_I =_k M \) and \( k\text{stut}(M_I) \). From Lemma 10, we get \( M'_I =_k M' \) and \( k\text{stut}(M'_I) \). Because \( 2\text{stut}(M) \), \( M_I =_k M \), and \( k\text{stut}(M_I) \), we have \( 2\text{stut}(M_I) \). From Lemma 9, we get \( 2\text{stut}(M'_I) \). From \( M'_I =_k M' \), we then get \( 2\text{stut}(M') \). \( \square \)
∃i: 1 ≤ i ≤ k: isSimple(if e then C_1 else C_2 end, M, i)

\[ M(e) \neq 0 \quad cc' = M(cc).push((M(T(e)), \emptyset, \emptyset)) \]

\[
\langle \text{if } e \text{ then } C_1 \text{ else } C_2 \text{ end}, M \rangle \rightarrow \langle C_1; i\text{-exit}, M[cc \mapsto cc'] \rangle
\]

∃i: 1 ≤ i ≤ k: isSimple(if e then C_1 else C_2 end, M, i)

\[ M(e) = 0 \quad cc' = M(cc).push((M(T(e)), \emptyset, \emptyset)) \]

\[
\langle \text{if } e \text{ then } C_1 \text{ else } C_2 \text{ end}, M \rangle \rightarrow \langle C_2; i\text{-exit}, M[cc \mapsto cc'] \rangle
\]

\[
\text{(Exit}_\text{IFS}) \quad \langle i\text{-exit}, M \rangle \rightarrow \langle \text{stop}, M[\forall j: i < j \leq k: T^j(w_i) \mapsto \bot, cc \mapsto cc'] \rangle
\]

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 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

\[ i > 1, M(T^{i-1}(e)) \neq \bot, \text{ and } M(T^i(e)) = \bot, \]

(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:

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

(21)

where anchor variable $m$ is associated with $\langle M, \bot, \bot, \bot \rangle$, anchor variable $h$ is associated with $\langle H, \bot, \bot, \bot \rangle$, and $h \neq 4$. Without considering the context (i.e., $m > 0$), flexible variable $w$ would be associated with either $\langle H, \bot, \bot, \bot \rangle$ (due to $w := h$) or $\langle \bot, \bot, \bot, \bot \rangle$ (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 $T(m) = M$. Thus, if $m > 0$, then $w$ is associated with $\langle H, M, \bot, \bot \rangle$, at the end of the conditional command. Otherwise, $w$ is associated with $\langle M, M, \bot, \bot \rangle$. Notice that, in both cases, the meta-meta label of $w$ is strictly less restrictive than its meta-label. So, using the meta-label to specify its own sensitivity would be conservative. In particular, using rules from $k$-$Enf$, $w$ would be associated with $\langle M, M, M, M \rangle$ or $\langle H, M, M, M \rangle$ 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:

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

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

$$\forall j \geq i: T^j(w_i) = \bot$$

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

$$\forall j > i: T^j(w_i) = \bot.$$ 

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

56
Consider now lattice $L_3 \triangleq \langle \{H, M, L\}, \sqsubseteq \rangle$ with $\bot = L \sqsubseteq M \sqsubseteq H$ and the following program:

\[
\begin{align*}
\text{if } m > 0 & \text{ then } w := h \text{ else } w := 4 \text{ end;} \\
\text{if } l > 0 & \text{ then } w' := w \text{ else } w := m \text{ end;}
\end{align*}
\]

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

For enforcer $k$-$E$opt, we have $n_{k\text{-}Eopt} = k + 1$, $\text{Aux}_{k\text{-}Eopt} = \{cc, bc\}$, $\text{Init}_{k\text{-}Eopt}(cc) = \epsilon$, and $\text{Init}_{k\text{-}Eopt}(bc) = \bot$.

## D Soundness of $k$-$E$opt

### Lemma 12. $k$-$E$opt 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 $k$-$E$opt. Also, we substitute (I\text{f}1) and (I\text{f}2) with:

\[
\begin{align*}
\tag{I\text{f}1'} \exists i: 1 \leq i \leq k: & \text{isSimple}\left(\text{if } e \text{ then } C_1 \text{ else } C_2 \text{ end}, M, i\right) \\
M(e) & \neq 0 \\
A & = \text{targetAnchor}(C_2) \\
cc' & = M(cc).\text{push}(\langle M(T(e)), W, A\rangle) \\
\langle \text{if } e \text{ then } C_1 \text{ else } C_2 \text{ end}, M \rangle & \rightarrow \langle C_1; \text{exit}, M[cc \mapsto cc'] \rangle
\end{align*}
\]

\[
\begin{align*}
\tag{I\text{f}2'} \exists i: 1 \leq i \leq k: & \text{isSimple}\left(\text{if } e \text{ then } C_1 \text{ else } C_2 \text{ end}, M, i\right) \\
M(e) & = 0 \\
A & = \text{targetAnchor}(C_1) \\
cc' & = M(cc).\text{push}(\langle M(T(e)), W, A\rangle) \\
\langle \text{if } e \text{ then } C_1 \text{ else } C_2 \text{ end}, M \rangle & \rightarrow \langle C_2; \text{exit}, M[cc \mapsto cc'] \rangle
\end{align*}
\]

It is easy to prove that $k$-$E$opt is an enforcer on $R$ and satisfies restrictions (E1), (E2), and (E3) by induction on the rules for $k$-$E$opt.\textsuperscript{9} We omit the details.

\textsuperscript{9}For $k$-$E$opt, we could use \texttt{exit} and introduce a new auxiliary for tracking when a simple \texttt{if} is executed. For simplicity, we instead introduce a new conditional delimiter $i$-\texttt{exit} and extend definition $\langle C, M \rangle = \triangledown (C', M')$ to hold even if the syntax of conditional delimiters that appear in $C$ and $C'$ is different.
We prove that BNI+(\(k\text{-Eopt,}\mathcal{L},C\)) holds, for a command \(C\) and a lattice \(\mathcal{L}\).
Assume \(\ell \in \mathcal{L}\) and
\[
M|_{\ell} = M'|_{\ell},
M(\text{cc}) = M'(\text{cc}), \text{mon}(M), \text{mon}(M')
\]
\[
\tau = \langle C, M \rangle \xrightarrow{*} \langle C_{\ell}, M_{\ell} \rangle \text{ according to } k\text{-Eopt},
\]
\[
\tau' = \langle C, M' \rangle \xrightarrow{*} \langle C'_{\ell}, M'_{\ell} \rangle \text{ according to } k\text{-Eopt}
\]
where \(C_{\ell}\) and \(C'_{\ell}\) are terminations (normal or blocked).

We prove \(c_1, c_2, c_3, \text{ and } c_4\). 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 \(k\text{-Eopt}\).

If \(C\) is \textbf{skip} or \(a := e\), then \(k\text{-Eopt}\) and \(\infty\text{-Enf}\) 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 \(k\text{-Eopt}\) and \(\infty\text{-Enf}\) use the same rules up to the \(k\)th tag. We follow the same proof as in Cases 3 of Lemma 1 by bounding \(r \leq k\) and recalling \(T^{k}(x) = T^{k+1}(x)\) (dy definition of \(k\text{-Eopt}\)).

If \(C\) is \(C_1; C_2\), \textbf{while } e \textbf{ do } C_1 \textbf{ end}, or \textbf{exit}, then \(k\text{-Eopt}\) and \(\infty\text{-Enf}\) use the same rules up to the \(k\)th 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 \(k\text{-Eopt}\)).

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

\[C\text{ is of the form if } a > 0 \text{ then } w_i := e_i \textbf{ else } w_i := n \textbf{ end,} \quad (22)\]
\[a \text{ is an anchor variable,} \quad (23)\]
\[w_i \text{ is a flexible variable,} \quad (24)\]
\[i = 1 \text{ and } M(T(e_i)) = \bot, \text{ or} \quad (25)\]
\[i > 1 \text{ and } M(T^{i-1}(e_i)) \neq \bot \text{ and } M(T^i(e_i)) = \bot \quad (26)\]
\[n \text{ is a constant,} \quad (27)\]
\[C\text{ is context-free (e.g., } M(\text{cc}) = \epsilon \text{ and } M(\text{bc}) = \bot). \quad (28)\]

- From (25), \(\text{mon}(M), \text{ and } M|_{\ell} = M'|_{\ell}, \) 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,
\[ i = 1 \text{ and } M'(T(e_i)) = \bot, \text{ or } \]
\[ i > 1 \text{ and } M'(T^{i-1}(e_i)) \neq \bot \text{ and } M'(T^i(e_i)) = \bot \]  
(28)

- From (27), we have \( M(cc) = \epsilon \). From \( M(cc) = M'(cc) \), we then get
\[ M'(cc) = \epsilon. \]  
(29)

- From (27), we have \( M(bc) = \bot \). From \( M|_\ell = M'|_\ell \), we then get
\[ M'(bc) = \bot. \]  
(30)

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

So, it remains to handle the case where \( \tau \) and \( \tau' \) 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, \( c_2, c_3, \) and \( c_4 \) are trivially true.

We prove \( c_1 \). Assume \( C \) is if \( a > 0 \) then \( w_i := e_i \) else \( w_i := n \) end and
\[ M|_\ell = M'|_\ell, \]
\[ M(cc) = M'(cc), \text{ mon}(M), \text{ mon}(M') \]
\[ \tau = \langle C, M \rangle \rightarrow \langle C_b; i\text{-exit}, M_b \rangle \rightarrow \langle i\text{-exit}, M_e \rangle \rightarrow \langle \text{stop}, M_i \rangle, \]
\[ \tau' = \langle C, M' \rangle \rightarrow \langle C'_b; i\text{-exit}, M'_b \rangle \rightarrow \langle i\text{-exit}, M'_e \rangle \rightarrow \langle \text{stop}, M'_i \rangle \]

where \( C_b \) and \( C'_b \) are either \( w_i := e_i \) or \( w_i := n \).

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

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 \( |cc| = \bot \) and \( bc = \bot \) at the beginning of its execution.
(I) After execution of $w_i := e_i$:
From $\text{ASGNF}$, $\text{IFS}$, and (25) we have:
\[ \forall j: 1 \leq j < i; T^j(w_i) = T^j(e_i) \sqcup T(a) \text{ and } \forall j: i \leq j \leq k; T^j(w_i) = T(a). \]

(II) After execution of $w_i := n$:
From $\text{ASGNF}$ and $\text{IFS}$ 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 $\text{EXIT}$, $\text{IFS}$ and (I) we have:
\[ \forall j: 1 \leq j < i; T^j(w_i) = T^j(e_i) \sqcup T(a) \text{ and } T^i(w_i) = T(a) \text{ and } \forall j: i < j \leq k; T^j(w_i) = \bot. \]

(IV) After execution of $i$-exit when $a \neq 0$ holds:
From $\text{EXIT}$, $\text{IFS}$ and (II) we have:
\[ \forall j: 1 \leq j \leq i; T^j(w_i) = T(a) \text{ and } \forall j: i < j \leq k; T^j(w_i) = \bot. \]

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 $\tau|_\ell = \text{obs}\, \tau'|_\ell$ and $M|_\ell = M'|_\ell$.

1. $M(T(a)) \sqsubseteq \ell$:
From $M|_\ell = M'|_\ell$, we get $M(a) = M'(a)$. So, $\tau$ and $\tau'$ take the same branch. We first prove $\tau|_\ell = \text{obs}\, \tau'|_\ell$. 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 $j = i$. $M_e(T^i(w_i)) \sqsubseteq \ell$ and $M'_e(T^i(w_i)) \sqsubseteq \ell$ hold because from (I), we have $M_e(T^i(w_i)) = M'_e(T^i(w_i)) = M(T(a))$ 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 $M_e(T^{j-1}(w_i)) = M(T^{i-1}(e_i)) \sqcup M(T(a)) = M'(T^{i-1}(e_i)) \sqcup M'(T(a)) = M'_e(T^{j-1}(w_i))$. 

60
Consider \( j < i \). Assume \( M_\ell(T_j(w_i)) \subseteq \ell \). Then, from (I), we have \( M(T_j(e_i)) \subseteq \ell \). 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 \( M_\ell(T_j^{-1}(w_i)) = M(T_j^{-1}(e_i)) \sqcup M(T(a)) = M'(T_j^{-1}(e_i)) \sqcup M'(T(a)) = M'_\ell(T_j^{-1}(w_i)). \)

- **Branch \( w_i := e_i \) is executed.**
  From (II) and because \( T^{k+1}(w_i) = T^k(w_i) \), we get 
  \[
  \forall j: 1 \leq j \leq k + 1: M_\ell(T_j^i(w_i)) = M'_\ell(T_j^i(w_i)) = M(T(a)).
  \]
  Also, \( M_\ell(w_i) = M'_\ell(w_i) = n. \)

So, \( \tau|_\ell = \text{obs} \tau'|_\ell \) holds.

We now prove \( M_{t|_\ell} = M'_{t|_\ell}. \) Because \( M_\ell(cc) = M'_\ell(cc) \) and because bc is not modified, it suffices to prove \( M|_\ell = M'|_\ell \) for the label chain of \( w_i \). We prove for \( j \) with \( 1 \leq j \leq k + 1 \) that if \( M_\ell(T_j^i(w_i)) \subseteq \ell \), then \( M'_\ell(T_j^i(w_i)) \subseteq \ell \) and \( M_\ell(T_j^{-1}(w_i)) = M'_\ell(T_j^{-1}(w_i)) \). 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 
  \[
  \forall j: i \leq j \leq k + 1: M_\ell(T_j^i(w_i)) = M'_\ell(T_j^i(w_i)).
  \]
  So, it suffices to examine \( j < i \).

  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 \( M_\ell(T_j^i(w_i)) = M(T^{i-1}(e_i)) \sqcup M(T(a)) = M'_\ell(T_j^i(w_i)) \sqcup M'(T(a)) = M'_\ell(T_j^{-1}(w_i)). \)

  Consider \( j < i \). Assume \( M_\ell(T_j^i(w_i)) \subseteq \ell \). Then, from (III), we have 
  \[
  M(T_j^i(e_i)) \subseteq \ell. \]
  From \( M|_\ell = M'|_\ell \), we then have 
  \[
  M(T_j^{-1}(w_i)) = M(T_j^{-1}(e_i)) \sqcup M(T(a)) = M'_\ell(T_j^{-1}(w_i)) \sqcup M'(T(a)) = M'_\ell(T_j^{-1}(w_i)).
  \]

- **Branch \( w_i := n \) is executed.**
  From (IV) and because \( T^{k+1}(w_i) = T^k(w_i) \), we get 
  \[
  \forall j: 1 \leq j \leq k + 1: M_\ell(T_j^i(w_i)) = M'_\ell(T_j^i(w_i)).
  \]
  Also, \( M_\ell(w_i) = M'_\ell(w_i) = n. \)

Thus, \( M_{t|_\ell} = M'_{t|_\ell} \) holds.

2. \( M(T(a)) \not\subseteq \ell \):
   Traces \( \tau \) and \( \tau' \) may take different branches. From (I), (II), \( M(T(a)) \not\subseteq \ell \),
and because $T^{k+1}(w_i) = T^k(w_i)$, we get that:
\[ \forall j: 1 \leq j \leq k + 1: M_\epsilon(T^j(w_i)) \not\subseteq \ell \land M'_\epsilon(T^j(w_i)) \not\subseteq \ell. \]
Thus, $\tau|_\ell =_{\text{obs}} \epsilon$ and $\tau'|_\ell =_{\text{obs}} \epsilon$.

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

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

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

### E Permissiveness of $k\text{-}\text{Enf}$ versus Chain Length

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

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

To prove $k\text{-}\text{Enf} \leq_{p_k}^L (k + 1)\text{-}\text{Enf}$, it suffices to also show that $(k + 1)\text{-}\text{Enf} \geq_{p_k}^L k\text{-}\text{Enf}$. Because $\mathcal{L}$ contains at least one non-bottom element $\ell$, with $\bot \subseteq \ell$, there exists $M_1$ such that $M_1 \models \mathcal{H}_0((k + 1)\text{-}\text{Enf}, \mathcal{L}, C)$, $\tau = \text{trace}_{(k+1)\text{-}\text{Enf}}(w := w_1, M_1) = \langle w := w_1, M_1 \rangle \rightarrow \langle \text{stop}, M_2 \rangle$, and $M_1(T^{k+1}(w_1)) \subseteq M_1(T^k(w_1))$.

There exists $M'_1$ such that $M'_1 \models \mathcal{H}_0(k\text{-}\text{Enf}, \mathcal{L}, C)$, $M'_1|_\ell = M'|_\ell$, and $\tau' = \text{trace}_{k\text{-}\text{Enf}}(w := w_1, M'_1) = \langle w := w_1, M'_1 \rangle \rightarrow \langle \text{stop}, M'_2 \rangle$.

From $M'_1|_\ell = M'|_\ell$, we have $M'_1(T^k(w_1)) = M_1(T^k(w_1))$. By definition of $k\text{-}\text{Enf}$ (i.e., $\forall x: T^{k+1}(x) = T^k(x)$), we then have...
Lemma 13. Consider step $\tau = \langle C, M_1 \rangle \rightarrow \langle C_2, M_2 \rangle$ generated by $k$-\text{Enf} for $k \geq 2$. Consider step $\tau' = \langle C, M'_1 \rangle \rightarrow \langle C'_2, M'_2 \rangle$ generated by $(k + 1)$-\text{Enf}. If $M_1 =_k M'_1$, then $C_2 = C'_2$, $M_2 =_k M'_2$, and $\tau_\ell |_{\ell} \leq \tau'_\ell |_{\ell}$, for all $\ell$.

Proof. By structural induction on $C$.

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

Proof. We prove $k$-\text{Enf} $\preceq_c^{k, \mathcal{L}} (k + 1)$-\text{Enf} and $(k + 1)$-\text{Enf} $\preceq_c^{k, \mathcal{L}} k$-\text{Enf}.

Consider conventionally initialized memory $M$ with $M \models \mathcal{H}_0(k$-\text{Enf}, $\mathcal{L}, C)$ and $\tau = \text{trace}_{k$-\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 $\tau' = \text{trace}_{(k + 1)$-\text{Enf}}(C, M')$. So, $M'$ is conventionally initialized, too. Thus, we have $\text{2stut}(M)$ and $\text{2stut}(M')$. 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 $\tau$ and Lemma 14, we get that $\tau_\ell |_{\ell} = \tau'_\ell |_{\ell}$, for all $\ell \in \mathcal{L}$. So, $k$-\text{Enf} $\preceq_c^{k, \mathcal{L}} (k + 1)$-\text{Enf} and $(k + 1)$-\text{Enf} $\preceq_c^{k, \mathcal{L}} k$-\text{Enf}. Thus, $k$-\text{Enf} $\preceq_c^{k, \mathcal{L}} (k + 1)$-\text{Enf}.

Lemma 14. Consider step $\tau = \langle C, M_1 \rangle \rightarrow \langle C_2, M_2 \rangle$ generated by $k$-\text{Enf} for $k \geq 2$. Consider step $\tau' = \langle C, M'_1 \rangle \rightarrow \langle C'_2, M'_2 \rangle$ generated by $(k + 1)$-\text{Enf}. If $M_1 =_k M'_1$, $\text{2stut}(M_1)$, and $\text{2stut}(M'_1)$, then $C_2 = C'_2$, $M_2 =_k M'_2$, $\text{2stut}(M_2)$, and $\tau_\ell |_{\ell} = \tau'_\ell |_{\ell}$, for all $\ell$.

Proof. By structural induction on $C$.

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

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

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

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

$M'_1(T^{k+1}(w_1)) = M'_1(T^k(w_1)) = M_1(T^k(w_1)) \triangleq M_1(T^{k+1}(w_1))$, and thus, we get $M'_2(T^{k+1}(w)) \triangleq M_2(T^{k+1}(w))$. So, $\tau$ generates observation involving $T^k(w)$ to label $M_2(T^{k+1}(w))$, but $\tau'$ does not generate observation involving $T^k(w)$ to label $M_2(T^{k+1}(w))$. So, $(k + 1)$-\text{Enf} $\not\preceq_{\rho_k}^{k, \mathcal{L}} k$-\text{Enf}. Thus, $k$-\text{Enf} $\preceq_{\rho_k}^{k, \mathcal{L}} (k + 1)$-\text{Enf}.


such that $M' \models \mathcal{H}_0(k\text{-}Enf, \mathcal{L}, C)$, $\rho_2(M, M')$, and $\tau' = \text{trace}_{k\text{-}Enf}(C, M')$. Using induction on the number of steps in $\tau$ and Lemma 16, we get that $\tau|^{0}_\ell = \tau'|^{0}_\ell$, for all $\ell \in \mathcal{L}$. So, $k\text{-}Enf \preceq_{p_2} 2\text{-}Enf$ and $2\text{-}Enf \preceq_{p_2} k\text{-}Enf$. Thus, $2\text{-}Enf \simeq_{p_2} k\text{-}Enf$. □

Lemma 16. Consider step $\tau = \langle C, M_1 \rangle \rightarrow \langle C_2, M_2 \rangle$ generated by $k\text{-}Enf$ for $k \geq 2$. Consider step $\tau' = \langle C, M'_1 \rangle \rightarrow \langle C'_2, M'_2 \rangle$ generated by $2\text{-}Enf$. 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 $\ell$.

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 $(k - 1)\text{-}BNI(\mathcal{L})$, 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, if $E \leq_{e}^{k - 1, \mathcal{L}} E'$, then $E'$ 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 $k\text{-}Eopt$ is an enforcer, satisfies $(k - 1)\text{-}BNI(\mathcal{L}_k)$, 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 $k\text{-}Eopt$.

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

$$\Omega = \langle \ell_1, \ell_2, \ldots, \ell_k \rangle$$

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 $E \leq_{e}^{k - 1, \mathcal{L}} E'$.

We prove that $E'$ does not satisfy $(k - 1)\text{-}BNI(\mathcal{L})$. Assume for contradiction that $E'$ satisfies $(k - 1)\text{-}BNI(\mathcal{L})$. We have that there are $j$, $C$, and $M \models \mathcal{H}_0(E, \mathcal{L}, C)$ such that $\Omega$ is $k$-precise at the $j$th state of $\tau = \text{trace}_{E}(C, M)$. 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 $\tau_1 = \text{trace}_{E'}(C, M_1)$. 64
By definition of \( k \)-precise and because \( E \leq_{E'}^{k-1} E' \) and \( E' \) satisfies \((k - 1)\)-BNI(\( \mathcal{L} \)), we then get that \( \tau_1 \) produces \( \Omega \) at the \( j \)th state. So, by definition, \( 1 \leq j \leq |\tau_1| \) and there exists \( w \) such that:

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

Working similarly for \( \Omega' \), we get:

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

for \( \tau_2 = \text{trace}_{E'}(C', M_2) \), 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 \( \forall i: 1 \leq i < k: M_r(T^i(w)) = M'_r(T^i(w')) = \ell_i \), 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.

**Lemma 17.** For \( k \geq 2 \), \( k\text{-Eopt} \) 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 \( k\text{-Eopt} \) is an enforcer on \( R \) for \( k\text{-BNI} \). Thus, \( k\text{-Eopt} \) satisfies \((k - 1)\)-BNI(\( \mathcal{L}_k \)).

Lemma 18 gives the possible label chains that \( k\text{-Eopt} \) 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 \( \langle \ell_{k-1}, \ell_{k-1}, \ldots, \ell_{k-1}, \bot \rangle \) 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:

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

where all $w_k$ and $z_k$ are flexible variables. Assume lattice $L_k$ of labels such that

\[
\ell_0 \sqsubseteq \ell_1 \sqsubseteq \ell_2 \sqsubseteq \ldots \sqsubseteq \ell_k \sqsubseteq \perp \tag{31}
\]

Assume $L_k$ consists only of $\perp, \ell_j$, 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 $k \geq 2$.

(Z$_1$) After the execution of $z_1 := w_1$, this is the possible chain for $z_1$:

\[
\begin{align*}
T(z_1) & \quad T^2(z_1) \quad \ldots \quad T^k(z_1) \\
\langle \ell_1 & \quad \perp \quad \ldots \quad \perp \rangle
\end{align*}
\]

(Z$_2$) After the execution of $z_2 := w_2$, these are the possible 2 chains for $z_2$:

\[
\begin{align*}
T(z_2) & \quad T^2(z_2) \quad T^3(z_2) \quad \ldots \\
\langle \ell_1 & \quad \ell_2 \quad \perp \quad \ldots \rangle & \quad a_2 > 0 \\
\langle \ell_2 & \quad \ell_2 \quad \perp \quad \ldots \rangle & \quad a_2 \neq 0
\end{align*}
\]

(Z$_3$) After the execution of $z_3 := w_3$, these are the possible 3 chains for $z_3$:

\[
\begin{align*}
T(z_3) & \quad T^2(z_3) \quad T^3(z_3) \quad T^4(z_3) \quad \ldots \\
\langle \ell_1 & \quad \ell_2 \quad \ell_3 \quad \perp \quad \ldots \rangle & \quad a_2 > 0 \land a_3 > 0 \\
\langle \ell_2 & \quad \ell_2 \quad \ell_3 \quad \perp \quad \ldots \rangle & \quad a_2 \neq 0 \land a_3 > 0 \\
\langle \ell_3 & \quad \ell_3 \quad \ell_3 \quad \perp \quad \ldots \rangle & \quad a_3 \neq 0
\end{align*}
\]
(Z_j) After the execution of \( z_j \coloneqq w_j \), these are the possible \( j \) chains for \( z_j \):

\[
\begin{array}{cccccc}
T & T^2 & T^3 & \ldots & T^{j-1} & T^j & T^{j+1} & \cdots \\
\langle \ell_1, \ell_2, \ell_3, \ldots, \ell_{j-1}, \ell_j, \bot \ldots \rangle & a_2, a_3, \ldots, a_j > 0 \\
\langle \ell_2, \ell_2, \ell_3, \ldots, \ell_{j-1}, \ell_j, \bot \ldots \rangle & a_2 \neq 0 \wedge a_3, \ldots, a_j > 0 \\
\langle \ell_3, \ell_3, \ell_3, \ldots, \ell_{j-1}, \ell_j, \bot \ldots \rangle & a_3 \neq 0 \wedge a_4, \ldots, a_j > 0 \\
\cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\
\langle \ell_{j-1}, \ell_{j-1}, \ell_{j-1}, \ldots, \ell_{j-1}, \ell_j, \bot \ldots \rangle & a_{j-1} \neq 0 \wedge a_j > 0 \\
\langle \ell_j, \ell_j, \ell_j, \ldots, \ell_j, \bot \ldots \rangle & a_j \neq 0 \\
\end{array}
\]

(Z_k) After the execution of \( z_k \coloneqq w_k \), these are the possible \( k \) chains for \( z_i \):

\[
\begin{array}{cccccc}
T & T^2 & T^3 & \ldots & T^{k-1} & T^k \\
\langle \ell_1, \ell_2, \ell_3, \ldots, \ell_{k-1}, \ell_k \rangle & a_2 > 0 \wedge a_3 > 0 \wedge \ldots \wedge a_j > 0 \\
\langle \ell_2, \ell_2, \ell_3, \ldots, \ell_{k-1}, \ell_k \rangle & a_2 \neq 0 \wedge a_3 > 0 \wedge \ldots \wedge a_k > 0 \\
\langle \ell_3, \ell_3, \ell_3, \ldots, \ell_{k-1}, \ell_k \rangle & a_3 \neq 0 \wedge \ldots \wedge a_k > 0 \\
\cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\
\langle \ell_{k-1}, \ell_{k-1}, \ell_{k-1}, \ldots, \ell_{k-1}, \ell_k \rangle & a_{k-1} \neq 0 \wedge a_k > 0 \\
\langle \ell_k, \ell_k, \ell_k, \ldots, \ell_k, \ell_k \rangle & a_k \neq 0 \\
\end{array}
\]

Lemma 18. The label chains presented after \( \text{pgm}_k \) are the only possible chains that \( k\text{-Eopt} \) 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 \( C_1 \):

\[
\text{if } a_1 > 0 \text{ then } w_1 := 0 \text{ else } w_1 := 1 \text{ end}
\]

with a memory \( M' \), then \( \text{isSimple}(C_1, M', 1) \) is always satisfied. Using \( \text{ifs}, \text{exit ifs}, \) and \( \text{asnf} \) rules, we get that \( z_1 \) is always associated with: \( \langle \ell_1, \bot, \ldots, \bot \rangle \).

Induction case:

IH: chains presented after \( \text{pgm}_k \) for \( z_{j-1} \), with \( j > 1 \), are the only possible chains that \( k\text{-Eopt} \) produces for \( z_{j-1} \).

We prove that chains presented after \( \text{pgm}_k \) are the only possible chains that \( k\text{-Eopt} \) produces for \( z_j \). When execution reaches \( C_j \):

\[
\text{if } a_j > 0 \text{ then } w_j := z_{j-1} \text{ else } w_j := j \text{ end}
\]

with some memory \( M' \), then using IH on \( z_{j-1} \) we get that \( \text{isSimple}(C_j, M', j) \) is always satisfied. So, rules \( \text{ifs} \) and \( \text{exit ifs} \) are used while executing \( C_j \).
1. \( a_j > 0 \)

Assignment \( w_j := z_{j-1} \) is executed. From IH, \( \text{IfS} \) and \( \text{AsgnF} \), we have that \( w_j \) is associated after \( w_j := z_{j-1} \) with one of the following \( j - 1 \) label chains:

\[
\begin{array}{c|c}
T & T^2 \; T^3 \; \ldots \; T^{j-1} \; T^j \; T^{j+1} \ldots \\
\langle \ell_1 \; \ell_2 \; \ell_3 \; \ldots \; \ell_{j-1} \; \ell_j \; \ell_j \ldots \rangle & a_2, a_3, \ldots, \\
& a_{j-1} > 0 \\
\langle \ell_2 \; \ell_2 \; \ell_3 \; \ldots \; \ell_{j-1} \; \ell_j \; \ell_j \ldots \rangle & a_2 \neq 0 \land a_3, \ldots, \\
& a_{j-1} > 0 \\
\langle \ell_3 \; \ell_3 \; \ell_3 \; \ldots \; \ell_{j-1} \; \ell_j \; \ell_j \ldots \rangle & a_3 \neq 0 \land a_4, \ldots, \\
& a_{j-1} > 0 \\
\ldots \\
\langle \ell_{j-1} \; \ell_{j-1} \; \ell_{j-1} \ldots \; \ell_{j-1} \; \ell_j \; \ell_j \ldots \rangle & a_{j-1} \neq 0 \\
\end{array}
\]

From \( \text{Exit}_\text{IfS} \), we have that \( w_j \) is associated at the end of \( C_j \) with one of the following \( j - 1 \) label chains:

\[
\begin{array}{c|c}
T & T^2 \; T^3 \; \ldots \; T^{j-1} \; T^j \; T^{j+1} \ldots \\
\langle \ell_1 \; \ell_2 \; \ell_3 \; \ldots \; \ell_{j-1} \; \ell_j \; \perp \ldots \rangle & a_2, a_3, \ldots, \\
& a_{j-1} > 0 \\
\langle \ell_2 \; \ell_2 \; \ell_3 \; \ldots \; \ell_{j-1} \; \ell_j \; \perp \ldots \rangle & a_2 \neq 0 \land \\
& a_3, \ldots, a_{j-1} > 0 \\
\langle \ell_3 \; \ell_3 \; \ell_3 \; \ldots \; \ell_{j-1} \; \ell_j \; \perp \ldots \rangle & a_3 \neq 0 \land \\
& a_4, \ldots, a_{j-1} > 0 \\
\ldots \\
\langle \ell_{j-1} \; \ell_{j-1} \; \ell_{j-1} \ldots \; \ell_{j-1} \; \ell_j \; \perp \ldots \rangle & a_{j-1} \neq 0 \\
\end{array}
\]

2. \( a_j \neq 0 \)

Assignment \( w_j := j \) is executed. From \( \text{IfS} \) and \( \text{AsgnF} \), we have that \( w_j \) is associated after \( w_j := j \) with the following label chain:

\[
\begin{array}{c|c}
T & T^2 \; T^3 \; \ldots \; T^{j-1} \; T^j \; T^{j+1} \ldots \\
\{ \ell_j \; \ell_j \; \ell_j \; \ldots \; \ell_j \; \ell_j \; \ell_j \ldots \} & a_2, a_3, \ldots, \\
& a_{j-1} > 0 \\
\end{array}
\]

From \( \text{Exit}_\text{IfS} \), we have that \( w_j \) is associated at the end of \( C_j \) with the following label chain:

\[
\begin{array}{c|c}
T & T^2 \; T^3 \; \ldots \; T^{j-1} \; T^j \; T^{j+1} \ldots \\
\{ \ell_j \; \ell_j \; \ell_j \; \ldots \; \ell_j \; \perp \ldots \} & a_2, a_3, \ldots, \\
& a_{j-1} > 0 \\
\end{array}
\]

68
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 $\tau = \text{trace}_{k\text{-Eopt}}(\text{pgm}_k, M)$, where $M$ is conventionally initialized and $M \models H_0(k\text{-Eopt}, \mathcal{L}_k, \text{pgm}_k)$. Then $\tau[s - 1] = (z_j := w_j; C, M_0)$ and $\tau[s] = (C, M_1)$ for some $1 < s \leq |\tau|$. Trace $\tau$ 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 $s$th state.

We prove that $\Omega$ is 1-precise. $\Omega$ may be one of the $j$ possible label chains in $(Z_j)$. 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 $i$th label chain in $(Z_j)$ has $T(z_j) = \ell_i$. So, it should be the case that

$$M(a_i) \neq 0, M(a_{i+1}) > 0, \ldots M(a_j) > 0 \quad \text{if } i \neq j$$

$$M(a_j) \neq 0 \quad \text{if } i = j$$

(32)

Consider an enforcer $D$ that satisfies 0-BNI($\mathcal{L}_k$) and $k$-Eopt $\leq^0_{c,\mathcal{L}_k} D$. Consider memory $M'$ with $M' \models H_0(D, \mathcal{L}_k, \text{pgm}_k)$, $\rho_1(M, M')$ and also $\tau' = \text{trace}_D(\text{pgm}_k, M')$. It should be the case that $\tau'[s] = (C, M'_1)$, because otherwise $\forall \ell \in \mathcal{L}_k: \tau'[^0_\ell] \not\subseteq \tau[0]_\ell$ (and $k$-Eopt $\leq^0_{c,\mathcal{L}_k} D$) 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 $M'_1(T(z_j)) \not\subsetneq M_1(T(z_j))$ or $M'_1(T(z_j)) \supsetneq M_1(T(z_j))$ holds. From $M'_1(T(z_j)) \not\subsetneq M_1(T(z_j))$ and $M'_1(T(z_j)) \not\supsetneq M_1(T(z_j))$, we get $\tau[0]_i \not\subseteq \tau'_[0]_i$, which implies that $k$-Eopt $\leq^0_{c,\mathcal{L}_k} D$, which is a contradiction. Assume $M_1(T(z_j)) \supsetneq M_1(T(z_j))$. From $M'_1(T(z_j)) \subsetneq M_1(T(z_j))$ and $M_1(T(z_j)) = \ell_i$, we then have $M'_1(T(z_j)) \subsetneq \ell_i$. Let $\ell = M'_1(T(z_j))$. There exists $M'''$ such that $M''' \models H_0(k$-Eopt, $\mathcal{L}_k, \text{pgm}_k)$ and

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

(33)

So $M'''$ is conventionally initialized.
Let $\tau'' = \text{trace}_{k-E\text{opt}}(pgm_k, M'')$. 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 $M'' \models H_0(D, L_k, pgm_k)$ and $\rho_1(M'', M'')$ hold.

Let $\tau'' = \text{trace}_D(pgm_k, M'')$. 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 $\forall \ell \in L_k: \tau''|_\ell^0 \leq \tau''|_\ell^0$ (and $k-\text{Eopt} \subseteq_0 L_k D$) 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

\[
M'(a_i) \neq 0, M'(a_{i+1}) > 0, \ldots M'(a_j) > 0 \text{ if } i \neq j
\]
\[
M'(a_j) \neq 0 \text{ if } i = j
\]

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

\[
M''(a_i) > 0, M''(a_{i+1}) > 0, \ldots M''(a_j) > 0 \text{ if } i \neq j
\]
\[
M''(a_j) > 0 \text{ if } i = j
\]

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

Because $M'_i(T(z_j)) = \ell$, observation $\langle z_j, M'_i(z_j) \rangle$ appears in $\tau'_0$. If $M'_i(z_j) \not\subseteq \ell$, then observation $\langle z_j, M'_i(z_j) \rangle$ does not appear in $\tau''|_{\ell}^0$, and thus $D$ does not satisfy 0-BNI($L_k$). If $M''_i(z_j) \not\subseteq \ell$, then observation $\langle z_j, M''_i(z_j) \rangle$ appears in $\tau''|_{\ell}^0$. But because $M'_i(z_j) \neq M''_i(z_j)$, $D$ does not satisfy 0-BNI($L_k$). So, in any case $D$ does not satisfy 0-BNI($L_k$), which is a contradiction. So, $M'_i(T(z_j)) = M_i(T(z_j))$. Thus, the label chain produced by $k-E\text{opt}$ for $z_j$ is 1-precise.

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

Assume $1 \leq n < j$. Consider $\tau = \text{trace}_{k-E\text{opt}}(pgm_k, M)$ for a conventionally initialized memory $M$ with $M \models H_0(k-\text{Eopt}, L_k, pgm_k)$. 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 $\tau$ 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.

70
We prove that $\Omega$ is $(n+1)$-precise. $\Omega$ may be one of the $j$ possible label chains in $(Z_j)$. So, we get $M_1(T^{n+1}(z_i)) = \ell_m$, for $1 \leq m \leq j$. Consider an enforcer $D$ that satisfies $n\text{-BNI}(L_k)$ and $k\text{-Eopt} \leq_{n,L_k} D$. Assume $\tau' = \text{trace}_D(pgm_k, M')$, for memory $M'$ with $M' \models \mathcal{H}_0(D, L_k, pgm_k)$ 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 $k\text{-Eopt} \leq_{n,L_k} D$ would not hold. From IH, $n\text{-BNI}(L_k, D)$, and $k\text{-Eopt} \leq_{n,L_k} D$ we get that

$$\forall t: 1 \leq t \leq n: M_1'(T^t(z_j)) = M_1(T^t(z_j)).$$  

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 $M_1'(T^{n+1}(z_j)) \not\in M_1(T^{n+1}(z_j))$ or $M_1'(T^{n+1}(z_j)) \sqsubseteq M_1(T^{n+1}(z_j))$ holds. From $M_1'(T^{n+1}(z_j)) \not\in M_1(T^{n+1}(z_j))$ and $M_1'(T^{n+1}(z_j)) \neq M_1(T^{n+1}(z_j))$, we get $k\text{-Eopt} \not\in_{n,L_k} D$, which is a contradiction.

Assume $M_1'(T^{n+1}(z_j)) \sqsubseteq M_1(T^{n+1}(z_j))$. So $M_1'(T^{n+1}(z_j)) \sqsubseteq \ell_m$. Let $\ell = M_1'(T^{n+1}(z_j))$. There exists memory $M_m$ such that $M_m \models \mathcal{H}_0(k\text{-Eopt}, L_k, pgm_k)$ and $M_m = M[-(M(a_m) > 0)]$, 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 $\tau_m = \text{trace}_{k\text{-Eopt}}(pgm_k, M_m)$. 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

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

There exists $M''$ with $M'' \models \mathcal{H}_0(D, L_k, pgm_k)$, $\rho_1(M_m, M'')$, and $\tau'' = \text{trace}_D(pgm_k, M'')$. 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 $\forall \ell \in L_k: \tau_m'|\ell \leq \tau''|\ell$ (and $k\text{-Eopt} \leq_{n,L_k} D$) would not hold. From $\rho_1(M_m, M'')$, $\rho_1(M, M')$, and $M_m = M[-(M(a_m) > 0)]$, 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 $\ell_m \not\in \ell$, we then get $M'|\ell = M''|\ell$. By IH, $n\text{-BNI}(L_k, D)$, and $k\text{-Eopt} \leq_{n,L_k} D$ 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 $\tau'_n|\ell$. If $M''(T^{n+1}(z_j)) \not\in \ell$, then observation $\langle T^n(z_j), M''(T^n(z_j)) \rangle$ does not appear in $\tau''|\ell$, and thus $n\text{-BNI}(L_k, D)$ does not hold. If $M''(T^{n+1}(z_j)) \sqsubseteq \ell$, then observation $\langle T^n(z_j), M''(T^n(z_j)) \rangle$ appears in $\tau''|\ell$. But because we have
Lemma 20. Consider $\tau = \text{trace}_{k\text{-Eopt}}(\text{pgm}_k, M)$, 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 $\tau' = \text{trace}_{k\text{-Eopt}}(\text{pgm}_k, M')$ 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. $M_1(T^n(z_j)) = \ell_m$

From (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 $M(a_m) \neq 0 \land M(a_{m+1}) > 0 \land \ldots \land M(a_j) > 0$. So, for $M'$ we should have $M'(a_m) > 0 \land M'(a_{m+1}) > 0 \land \ldots \land M'(a_j) > 0$, 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 (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. $M_1(T^n(z_j)) \neq \ell_m$

From (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 $M'(a_m) > 0 \land M'(a_{m+1}) > 0 \land \ldots \land M'(a_j) > 0$. So, $M'$ should have $M'(a_m) \neq 0 \land M'(a_{m+1}) > 0 \land \ldots \land M'(a_j) > 0$, because $M' = M[\neg(M(a_m > 0))]$. Thus, $M'_1(\Omega_{z_j})$ is the $m$th label chain in (Z$_j$). So, $M'_1(T^n(z_j)) = M'_1(T^{m-1}(z_j)) = \ell_m$. Thus, $M_1(T^n(z_j)) \neq M_1'(T^n(z_j))$.

\section*{F.2 Weakened Threat Model}

**Theorem 7.** For an enforcer $E$ and lattice $\mathcal{L}_3$, if $G_{n=e}^E \leq_{c}^{0,\mathcal{L}_3} E$, then $E$ does not satisfy $0\text{-BNI}(\mathcal{L}_3)$.

Proof. We have $\mathcal{L} = \langle \{L,M,H\}, \sqsubseteq \rangle$ where $L \sqsubseteq M \sqsubseteq H$ and $\bot = L$.
Consider program \textit{pgm}:
\[
\begin{align*}
w_a & := m_a; \\
\text{if } m_b > 0 \text{ then } w_b & := m_b \text{ else } w_b := h \text{ end}; \\
\text{if } l > 1 \text{ then } w_c & := w_a \text{ else } w_c := w_b \text{ end}; \\
w & := w_c; \\
m & := w; \\
l & := 1
\end{align*}
\]

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

\textit{2-Enf} produces the following labels when executes \textit{pgm} with a conventionally initialized memory:

\[
\begin{array}{|c|c|c|}
\hline
 & m_b > 0 & m_b \not> 0 \\
\hline
 l > 1 & w_a : \langle M, L \rangle & w_a : \langle M, L \rangle \\
 & w_b : \langle M, M \rangle & w_b : \langle H, M \rangle \\
 & w : \langle M, L \rangle & w : \langle M, L \rangle \\
\hline
 l \not> 1 & w_a : \langle M, L \rangle & w_a : \langle M, L \rangle \\
 & w_b : \langle M, M \rangle & w_b : \langle H, M \rangle \\
 & w : \langle M, M \rangle & w : \langle H, M \rangle \\
\hline
\end{array}
\]

We first prove that, for all executions, \( T(w) \) produced by \textit{2-Enf} is 1-precise.

Consider \( \tau = \text{trace}_{2-\text{Enf}}(\text{pgm}, M) \), where \( M \) is conventionally initialized and \( M \models \mathcal{H}_0(2-\text{Enf}, \mathcal{L}, \text{pgm}) \). 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 \( 2-\text{Enf} \leq^{0, \mathcal{L}} E' \). Assume \( \tau' = \text{trace}_{E'}(\text{pgm}, M') \) where \( M' \models \mathcal{H}_0(E', \mathcal{L}, \text{pgm}) \) and \( \rho_1(M, M') \).

From 2-\text{Enf} \( \leq^{0, \mathcal{L}} E' \), we get \( \forall \ell \in \mathcal{L}: \tau'_t^0 \preceq \tau'_t^0 \). 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 \( M_r(T(w)) = M \). We prove \( M'_r(T(w)) = M \). It should be the case
that $M'_r(T(w)) \subseteq M$, because otherwise $\tau^0_{|M} \subseteq \tau^0_{|M}$ would not hold, since observation $\langle w, M_r(w) \rangle$ would belong to $\tau^0_{|M}$, but no observation involving $w$ would belong to $\tau^0_{|M}$.

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

$$M'(l) \neq 1 \text{ and } M'(m_b) > 0 \quad (39)$$

There exists a memory $M''$ such that $M'' \models H_0(E', L, pgm)$ and:

$$M'|_L = M''|_L, \quad (40)$$
$$M''(m_b) \neq 0, \quad (41)$$
$$M'(m_b) \neq M''(h) \quad (42)$$

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

$$M'(l) = M''(l). \quad (43)$$

From $M'(l) \neq 1$, we then get

$$M''(l) \neq 1. \quad (44)$$

If $M''(T(w)) \neq L$, then $E'$ does not satisfy 0-BNI($\mathcal{L}$), because (40) and $M''(T(w)) = L$, and thus, observation $\langle w, M'_r(w) \rangle$ is included in $\tau''|_L$ but no observation involving $w$ is included in $\tau''|_L$. So, $M'_r(T(w)) = L$. Because $E'$ is an enforcer and due to (44), (39), and (41), we have:

$$M'_r(w) = M'_r(w_c) = M'_r(w_b) = M'(m_b)$$
$$M''(w) = M''(w_c) = M''(w_b) = M''(h).$$

From (42), we then have $M'_r(w) \neq M''(w)$. So, $E'$ does not satisfy 0-BNI given (40), because $\tau''|_L$ includes observation $\langle w, M'_r(w) \rangle$, $\tau''|_L$ includes observation $\langle w, M''(w) \rangle$, and $M'_r(w) \neq M''(w)$. But this is a contradiction. Thus, $M'_r(T(w)) = M$.

2. $M(l) \neq 1$ and $M(m_b) \neq 0$ (a2)

We have $M_r(T(w)) = H$. We prove that $M'_r(T(w)) = H$. If $M'_r(T(w)) = L$, then we follow the arguments of the above case, where (39) would be $M'(m_b) \neq 0$ and (41) would be $M''(m_b) > 0$, and we are lead to a contradiction.
Assume for contradiction that $M'_r(T(w)) = M$. There exists $M''$ such that $M'' \models H_0(E', \mathcal{L}, \text{pgm})$ and

\begin{align}
M'|_M &= M''|_M, \\
M'(h) &\neq M''(h). 
\end{align}  

(45)  
(46)

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

\begin{align}
M'(l) &= M''(l), \\
M'(m_b) &= M''(m_b). 
\end{align}  

(47)  
(48)

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

\begin{align}
M'(l) &\neq 1, M''(l) \neq 1, M'(m_b) \neq 0, \text{ and } M''(m_b) \neq 0. 
\end{align}  

(49)

It should be the case that $M''_r(T(w)) = M$, because otherwise $E'$ would not satisfy $0\text{-BNI}(\mathcal{L})$, which is a contradiction. Because $E'$ is an enforcer and due to (49), we have:

- $M'_r(w) = M'_r(w_c) = M'_r(w_b) = M'(h)$
- $M''_r(w) = M''_r(w_c) = M''_r(w_b) = M''(h)$.

From (46), we then have $M'_r(w) \neq M''_r(w)$. So, given (45), $E'$ does not satisfy $0\text{-BNI}(\mathcal{L})$, which is a contradiction. Thus, $M'_r(T(w)) = H$.

3. $M(l) > 1$ (a3)

We have $M_r(T(w)) = M$. We prove that $M'_r(T(w)) = M$. It should be the case that $M'_r(T(w)) \subseteq M$, because otherwise $\tau''|_M \subseteq \tau'|_M$ would not hold.

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

\begin{align}
M'|_L &= M''|_L, \\
M'(m_a) &\neq M''(m_a). 
\end{align}  

(50)  
(51)

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

\begin{align}
M'(l) &= M''(l). 
\end{align}  

(52)

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

\begin{align}
M'(l) > 1 \text{ and } M''(l) > 1. 
\end{align}  

(53)
If $M''(T(w)) \neq \mathbf{L}$, then $E'$ does not satisfy $0$-BNI($\mathcal{L}$), which is a contradiction.

Assume $M''(T(w)) = \mathbf{L}$. Because $E'$ is an enforcer and due to (53), we have

$M''(w) = M'(w_a) = M'(m_a)$ and

$M''(w) = M'(w_a) = M'(m_a)$.

From (51), we have $M'(w) \neq M''(w)$. So, given (50), $E'$ does not satisfy $0$-BNI($\mathcal{L}$), which is a contradiction. Thus, $M'(T(w)) = \mathbf{M}$.

So, for all executions, $T(w)$ produced by $2$-$\text{Enf}$ is 1-precise.

Consider an enforcer $E$ that uses 1-dependent $G^E_{t := 1}$ and $2$-$\text{Enf} \leq 0, \mathcal{L} E$. 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 $\text{pgm}$ for the following exhaustive list of cases: $l > 1$, $l \neq 1 \land m_a > 0$, and $l 
\neq 1 \land m_b 
\neq 0$. 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 $M \models \mathcal{H}_0(2$-$\text{Enf}, \mathcal{L}, \text{pgm})$ and $M(l) > 1$.

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

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

Because $M(l) > 1$, enforcer $2$-$\text{Enf}$ executes $l := 1$, and thus, $\langle l, 1 \rangle \in \tau'|_0$. From $\forall \ell: \tau'|_\ell \leq \tau_1|_\ell$, we then get $\langle l, 1 \rangle \in \tau_1|_0$. Thus, $\langle l := 1, M_r \rangle \rightarrow \langle \text{stop}, M_s \rangle$ should be a subtrace of $\tau_1$. Thus, we have

$M_r(G^E_{t := 1}) = \text{true}$.  \hspace{1cm} (55)

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

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

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

$M_r(T(l)) = \mathbf{L}, M_r(T(m)) = \mathbf{M}, M_r(T(w)) = \mathbf{M}$.  \hspace{1cm} (57)
2. \( l \neq 1 \) and \( m_b > 0 \):
There exists a conventionally initialized memory \( M' \) with 
\( M' \models \mathcal{H}_0(2\text{-}\text{Enf}, \mathcal{L}, \text{pgm}) \) \( M'(l) \neq 1 \) and \( M'(m_b) > 0 \).

Let \( \tau' = \text{trace}_{2\text{-}\text{Enf}}(\text{pgm}, M') \). There exists \( M_2 \) such that \( M_2 \models \mathcal{H}_0(E, \mathcal{L}, \text{pgm}) \), 
\( \rho_1(M', M_2) \), and \( \tau_2 = \text{trace}_E(\text{pgm}, M_2) \). Because \( 2\text{-}\text{Enf} \leq_{e,L} E \), we have that \( \forall \ell \in \mathcal{L}: \tau'_0[\ell] \leq \tau_2[\ell]_0 \). From \( \rho_1(M', M_2) \), we have

\[
M_2(l) \neq 1 \land M_2(m_b) > 0. \quad (59)
\]

Because \( M'(l) \neq 1 \), and \( M'(m_b) > 0 \), enforcer \( 2\text{-}\text{Enf} \) executes \( m := w \), and thus, \( \langle m, \nu \rangle \in \tau'_0[\ell] \). From \( \forall \ell: \tau'_0[\ell] \leq \tau_2[\ell]_0 \), we then get \( \langle m, \nu \rangle \in \tau_2[\ell]_0 \).

Thus, \( \langle m := w; l := 1, M_m \rangle \to \langle l := 1, M_r \rangle \) should be a subtrace of \( \tau_2 \).
Because \( \text{pgm} \) 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

\[
M_r(G_{l=1}^E) = f(M_r(T(l)), M_r(T(m)), M_r(T(w))). \quad (60)
\]

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

\[
M_r(T(l)) = \mathbf{L}, M_r(T(m)) = \mathbf{M}, M_r(T(w)) = \mathbf{M}. \quad (61)
\]

So, \( M_r(G_{l=1}^E) = f(L, M, M) = \text{true} \), due to (58). So, \( E \) executes \( l := 1 \), and thus, \( \langle l, 1 \rangle \in \tau_2[\ell]_0 \). (c1)

3. \( l \neq 1 \) and \( m_b \neq 0 \):
There exists a conventionally initialized memory \( M'' \) with 
\( M'' \models \mathcal{H}_0(2\text{-}\text{Enf}, \mathcal{L}, \text{pgm}) \), \( \text{dom}(M'') = \text{dom}(M') \), \( M''|L = M'|L \), \( M''(l) \neq 1 \), and \( M''(m_b) \neq 0 \).

Let \( \tau'' = \text{trace}_{2\text{-}\text{Enf}}(\text{pgm}, M'') \). There exists \( M_3 \) such that \( M_3 \models \mathcal{H}_0(E, \mathcal{L}, \text{pgm}) \), 
\( \rho_1(M'', M_3) \), and \( \tau_3 = \text{trace}_E(\text{pgm}, M_3) \). Because \( 2\text{-}\text{Enf} \leq_{e,L} E \), we have that \( \forall \ell \in \mathcal{L}: \tau''[\ell]_0 \leq \tau_3[\ell]_0 \). From \( \rho_1(M'', M_3) \), we have

\[
M_3(l) \neq 1 \land M_3(m_b) \neq 0. \quad (62)
\]
From \( \text{dom}(M'') = \text{dom}(M') \), \( M'|L = M''|L, \rho_1(M'', M_3), c(M''), \rho_1(M', M_2), c(M') \), we then get \( M_2|L = M_3|L \). Enforcer 2-\text{Enf} executes \( w := w_c \), and thus, \( \langle w, \nu \rangle \in \tau''|_0 H \). From \( \forall \ell: \tau''|_\ell \preceq \tau_3|_\ell \), we then get \( \langle w, \nu \rangle \in \tau_3|_H \). Thus, \( \langle w := w_c; m := w; l := 1, M_w \rangle \rightarrow \langle m := w; l := 1, M_m \rangle \) should be a subtrace of \( \tau_3 \). We examine two cases: \( \tau_3 \) either executes \( m := w \) or not.

3.1. \( \tau_3 \) does not execute \( m := w \).

So, \( l := 1 \) is not executed either. (c2)

Thus, \( \tau_2|_L \neq \tau_3|_L \), due to (c1) and (c2).

From \( M_2|L = M_3|L \), we then get that \( E \) does not satisfy 0-BNI(\( L \)).

3.2. \( \tau_3 \) executes \( m := w \).

Then \( h \) is leaked to \( m \). There exists \( M_4 \) such that \( M_4|L = H_0(E, L, pgm) \) and

\[
M_3|L = M_4|L, \quad (63)
\]
\[
M_3(h) \neq M_4(h). \quad (64)
\]

From (63) and (62), we get

\[
M_3(l) = M_4(l) \neq 1, \quad (65)
\]
\[
M_3(m_b) = M_4(m_b) \neq 0. \quad (66)
\]

Let \( \tau_4 = \text{trace}_E(pgm, M_4) \). If \( \tau_4 \) does not execute \( m := w \), then \( \tau_3|_M \neq \tau_4|_M \). If \( \tau_4 \) executes \( m := w \), then (64) implies \( \tau_3|_M \neq \tau_4|_M \), because in both traces \( \tau_3 \) and \( \tau_4 \) the value of \( m \) equals the value of \( h \). So, in both cases, \( E \) does not satisfy 0-BNI(\( L \)).

\[ \square \]

**Familiar Two-level Lattice**

**Theorem 8.** Enforcer \( E_{H,L} \) uses 1-dependent \( G_{a:e} \), satisfies 0-BNI(\( L_2 \)), and satisfies 2-\text{Enf} \( <_{H,L}^0 \) \( E_{H,L} \).

**Proof.** Lemma 24 gives that \( E_{H,L} \) is an enforcer and uses 1-dependent \( G_{a:e} \). Lemma 21 gives that \( E_{H,L} \) satisfies 0-BNI(\( L_2 \)). Lemma 22 gives that 2-\text{Enf} \( <_{H,L}^0 \) \( E_{H,L} \) holds. \( \square \)

**Lemma 21.** \( E_{H,L} \) satisfies 0-BNI(\( L_2 \)).

78
Proof. We retrieve $E_{H,L}$ from $\infty$-Enf by replacing rules for assignments and function $U$ used in exit with the corresponding definitions in Figure 11. Because BNI+($\infty$-Enf,$L_2$,$C$) holds, then BNI+($E_{H,L}$,$L_2$,$C$) 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+($E_{H,L}$,$L_2$,$C$) where $C$ is an assignment or exit.

For $E_{H,L}$, the domain of memories contain only variables and tags of these variables. So, in the definition of BNI+, projections $M|_\ell$ and $\tau|_\ell$ equal projection $M|_0$ and $\tau|_0$, correspondingly. Also, $\text{mon}(M)$ is trivially true for any such memory So, the definition on BNI+ becomes: For all $\ell \in L_2$, $M, M'$ if

$$M|_\ell = M'|_\ell,$$
$$M(\text{cc}) = M'(\text{cc})$$
$$\tau = \text{trace}_{E_{H,L}}(C, M) = \langle C, M \rangle \rightarrow \langle C_t, M_t \rangle,$$
$$\tau' = \text{trace}_{E_{H,L}}(C, M') = \langle C, M' \rangle \rightarrow \langle C'_t, M'_t \rangle$$

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 $\tau|_\ell = \text{obs} \tau'|_\ell$, $M_t|_\ell = M'_t|_\ell$, and $M_t(\text{cc}) = M'_t(\text{cc})$.

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

79
c3 If $C_t$ is stop, $C_t'$ is block, and $M'_t(bc) \not\subseteq \ell$, then $M_t(bc) \not\subseteq \ell$.

c4 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 $\tau'$, and $M_{tp}'([cc]) \subseteq \ell$, then there exists $\langle C'', M'' \rangle \in \tau$, with $C'' = C_{tp}'$ and $M''|_\ell = M_{tp}'|_\ell$.

Because $\ell \in L_2$, we have that $\ell$ is either $L$ or $H$. If $\ell = H$, then BNI+ is trivially satisfied, because hypothesis $M'|_\ell = M''|_\ell$ implies $M = M'$. Thus, we prove BNI+ for

$$\ell = L. \quad (67)$$

1. $C$ is $a := e$:

We prove that $M(T(a)) = M'(T(a))$. If $M(T(a)) = L$, then $M|_\ell = M'|_\ell$ gives $M'(T(a)) = L$. If $M(T(a)) = H$, then $M|_\ell = M'|_\ell$ gives $M'(T(a)) = H$. So, $M(T(a)) = M'(T(a))$.

1.1. $M(T(a)) \subseteq \ell$

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, $M(T(e)) \cup M([cc]) \cup M(bc) \subseteq M(T(a))$ holds. This implies that $M(T(e)) \subseteq \ell$, $M([cc]) \subseteq \ell$, and $M(bc) \subseteq \ell$. Because $M|_\ell = M'|_\ell$, we get $M(cc) = M'(cc)$ and $M(bc) = M'(bc)$. From $M(T(e)) \subseteq \ell$ and $M|_\ell = M'|_\ell$, we then get $M'(T(e)) \subseteq \ell$. From (67), we then have $M(T(e)) = M'(T(e)) = L$. Thus, $M'(T(e)) \cup M'( [cc] ) \cup M' (bc) \subseteq M'(T(a))$ 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:

$$\tau = \langle a := e, \mathcal{M} \rangle \rightarrow \langle \text{stop}, M[a \mapsto M(e), bc \mapsto \ell_g'] \rangle$$
$$\tau' = \langle a := e, \mathcal{M}' \rangle \rightarrow \langle \text{stop}, M'[a \mapsto M'(e), bc \mapsto \ell_g''] \rangle,$$

where $\ell_g = M([cc]) \cup M(bc)$ and $\ell_g' = M'([cc]) \cup M'(bc)$. 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.

\[ \tau = \langle a := e, M \rangle \rightarrow \langle \text{block}, M[bc \mapsto \ell_g] \rangle \]
\[ \tau' = \langle a := e, M' \rangle \rightarrow \langle \text{block}, M'[bc \mapsto \ell'_g] \rangle \]

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

1.2. \( M(T(a)) \nsubseteq \ell \)

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

We prove c1. Assume \( C_t = C'_t = \text{stop} \). Because \( M(T(a)) \), \( M'(T(a)) \nsubseteq \ell \), \( M_t \) and \( M'_t \) do not need to agree on \( a \). Assume \( M_t(bc) \subseteq \ell \). Then \( 2 \) holds.

We prove c3. Assume \( C_t \) is \text{stop}, \( C'_t \) is \text{block}, and \( M'_t(bc) \nsubseteq \ell \). We prove \( M_t(bc) \nsubseteq \ell \). We prove the contrapositive. Assume \( M'_t(bc) \subseteq \ell \), then following the same arguments as above, we get \( M_t(bc) = M'_t(bc) \), and thus, \( M'_t(bc) \subseteq \ell \), as wanted.

We prove c4. Assume \( C_t \) is \text{stop}, \( C'_t \) is \text{block}, \( \langle C'_t, M'_t \rangle \rightarrow \langle C'_t, M' \rangle \) are the last two states of \( \tau' \), and \( M'_t([cc]) \subseteq \ell \). So, \( M'_t = M' \) and \( C'_t = a := e \). We have that \( \langle C''_t, M'' \rangle = \langle a := e, M \rangle \), which satisfies \( C''_t = C'_t \) and \( M''|_\ell = M'_t|_\ell \). Thus c4 holds.

2. \( C = w := e \)

\[ \tau = \langle w := e, M \rangle \rightarrow \langle \text{stop}, M \rangle \]
\[ \tau' = \langle w := e, M' \rangle \rightarrow \langle \text{stop}, M' \rangle \]

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. \( M_t(T(w)) \subseteq \ell \)

Then \( M(T(e)) \subseteq \ell \), \( M([cc]) \subseteq \ell \), and \( M(bc) \subseteq \ell \). 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. $M_t(T(w)) \not\subseteq \ell$

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

3. exit

$\tau = \langle \text{exit}, M \rangle \rightarrow \langle \text{stop}, M_t \rangle$

$\tau' = \langle \text{exit}, M' \rangle \rightarrow \langle \text{stop}, M'_t \rangle$.

c2, c3, c4 are trivially true.

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

3.1. $M_t([cc]) \cup M_t(bc) \not\subseteq \ell$ and $M(jj).\text{top.A} \neq \emptyset$.

Because $M(jj).\text{top.A} \neq \emptyset$, we have $M_t(bc) = M(bc) \cup M([cc])$. We have $M_t([cc]) \subseteq M([cc])$. We get $M_t([cc]) \cup M(bc) \subseteq M([cc]) \cup M(bc)$, which becomes $M_t([cc]) \cup M(bc) \subseteq M([cc]) \cup M(bc)$. We get $M_t([cc]) \cup M([cc]) \subseteq M([cc]) \cup M(bc)$. Thus, $M'_t([cc]) \cup M'_t(bc) \not\subseteq \ell$. So, $M_t([cc]) \cup M_t(bc) \not\subseteq \ell$.

From $M_t(bc) \not\subseteq \ell$ and $M'_t(bc) \not\subseteq \ell$, we get $M_t([cc]) \cup M_t(bc) \not\subseteq \ell$ and $M'_t([cc]) \cup M'_t(bc) \not\subseteq \ell$.

Only variables in W change their labels. Let $x \in M(jj).\text{top.W}$. Because $M([cc]) \cup M(bc) \not\subseteq \ell$, we have $M_t(T(x)) \not\subseteq \ell$. Because $M(jj) = M'(jj)$, we get $x \in M'(jj).\text{top.W}$, too. Because $M'(jj) \cup M'(bc) \not\subseteq \ell$, we have $M'_t(T(x)) \not\subseteq \ell$. So, $M_t|_\ell = M'_t|_\ell$. Thus c1 holds.

3.2. $M_t([cc]) \cup M_t(bc) \not\subseteq \ell$ and $M(jj).\text{top.A} = \emptyset$.

Because $M(jj).\text{top.A} = \emptyset$, we have $M_t(bc) = M(bc)$. We have $M_t([cc]) \subseteq M([cc])$. We get $M_t([cc]) \cup M(bc) \subseteq M([cc]) \cup M(bc)$, which becomes $M_t([cc]) \cup M_t(bc) \subseteq M([cc]) \cup M(bc)$. So, $M_t([cc]) \cup M(bc) \not\subseteq \ell$. Because $M_t|_\ell = M'_t|_\ell$, we also get $M'(cc) \cup M'(bc) \not\subseteq \ell$. $M(jj).\text{top.A} = \emptyset$ and $M(jj) = M'(jj)$ give $M'(jj).\text{top.A} = \emptyset$. So, $M'_t(bc) = M'(bc)$. 82
Assume $M_t(bc) \subseteq \ell$. Then $M(bc) \subseteq \ell$. From $M|_\ell = M'|_\ell$, we then get $M(bc) = M'(bc)$. Thus, $M_t(bc) = M'_t(bc)$.

We prove $M'_t([cc]) \cup M'_t(bc) \not\subseteq \ell$. Assume for contradiction that $M'_t([cc]) \cup M'_t(bc) \subseteq \ell$. Then $M'_t(bc) \subseteq \ell$. 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, $M'_t([cc]) \cup M'_t(bc) \not\subseteq \ell$.

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

3.3. $M_t([cc]) \cup M_t(bc) \subseteq \ell$

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

- Let $M([cc]) \cup M(bc) \subseteq \ell$.
  Let $x \in M(cc).top.W$. Because $M(cc) = M'(cc)$, we get $x \in M'(cc).top.W$. We have:
  $M_t(T(x)) \subseteq \ell \Rightarrow M(T(x)) \subseteq \ell \Rightarrow M(x) = M'(x)$ and $M'(T(x)) \subseteq \ell$.

  Because $M(cc) = M'(cc)$ and $M(bc) = M'(bc)$, we then have $M_t(x) = M'_t(x)$ and $M'_t(T(x)) \subseteq \ell$. Thus, $M_t|_\ell = M'_t|_\ell$. Thus c1 holds.

- Let $M([cc]) \cup M(bc) \not\subseteq \ell$.
  So, $M'(cc) \cup M'(bc) \not\subseteq \ell$. Let $x \in M(cc).top.W$. So, $M_t(T(x)) \not\subseteq \ell$.

  Because $M(cc) = M'(cc)$, we get $x \in M'(cc).top.V$, and thus $M'_t(T(x)) \not\subseteq \ell$. 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:

- $M_{tp}(bc) \subseteq \ell$ and $M_{tp}(cc) \subseteq \ell$
  From IH[c4] on $C_1$, there exists $(C_{1tp}, M'_1) \in \tau'_1$ such that $M_{tp}|_\ell = M'_1|_\ell$.
  So, $M_{tp}(bc) = M'_1(bc)$ and $M_{tp}(cc) = M'_1(cc)$. Because $\tau_1$ is blocked, we have $M_{tp}(T(e)) \cup M_{tp}(cc) \subseteq M_{tp}(T(a))$. Since the inequality is satisfied in $\tau'_1$, 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 \( M'(T(e)) \) and \( M_p(T(e)) \) should either be both L or both H. Thus, this case is no longer possible once a two-level lattice is considered.

Thus, BNI+\((E_{H,L}, L_2, C)\) holds. BNI+ implies 0-BNI. So, \( E_{H,L} \) satisfies 0-BNI. □

**Lemma 22.** \( 2\text{-Enf} \overset{\leq}{\prec} 0_{\mathcal{L}_2} E_{H,L} \)

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

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. \( \forall i \geq 0: M'([cc.pop^i]) \sqsubseteq M([cc.pop^i]) \land M'(cc.pop^i.top.A) = M(cc.pop^i.top.A) \land M'(cc.pop^i.top.W) = M(cc.pop^i.top.W) \)

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 \( \tau \) and Lemma 23, we get that \( \forall \ell \in \mathcal{L}_2: \tau|_{\ell}^0 \preceq \tau'|_{\ell}^0 \). So, \( 2\text{-Enf} \overset{\leq}{\prec} 0_{\mathcal{L}_2} E_{H,L} \).

Now, we prove \( E_{H,L} \not\overset{\leq}{\prec} 0_{\mathcal{L}_2} 2\text{-Enf} \). Consider program \( \text{pgm} \):

```plaintext
if \( h > 0 \)
  \( w := h \)
else
  \( w := l \)
end;
\( h := w \);
\( l := 1 \)
```

For every execution under 2-Enf we have:
Lemma 23. If \( \langle C_1, M_1 \rangle \rightarrow \langle C_2, M_2 \rangle \) under 2-\textit{Enf}, and \( \langle C_1, M'_1 \rangle \rightarrow \langle C'_2, M'_2 \rangle \) under \( E_{H,L} \), then \( C_2 = C'_2 \) and \( M'_1 \subseteq_e M_1 \) or \( C_2 = \text{block} \) and \( M'_2 \subseteq_e M_2 \) hold.

Proof. We use induction on \( C_1 \). Assume \( C_2 \neq \text{block} \). We prove \( C_2 = C'_2 \) and \( M'_2 \subseteq_e M_2 \).

1. \( C_1 \) is \( a := e \).
   We have \( C_2 = \text{stop} \). Then \( M_1(T(e)) \cup M_1([cc]) \cup M_1(bc) \subseteq M_1(T(a)) \) holds. Due to \( M'_1 \subseteq_e M_1 \), we then get \( M'_1(T(e)) \cup M'_1([cc]) \cup M'_1(bc) \subseteq M'_1(T(a)) \). So, \( C'_2 = \text{stop} \). Also, \( M_2(a) = M_1(e) = M'_1(e) = M'_2(a) \).
   Because \( M'_2(bc) = M'_1([cc]) \cup M'_1(bc) \) and \( M_2(bc) = M_1(T(e)) \cup M_1([cc]) \cup M_1(bc) \), hypothesis \( M'_1 \subseteq_e M_1 \) gives \( M'_2(bc) \subseteq M_2(bc) \). So, \( M'_2 \subseteq_e M_2 \).

2. \( C_1 \) is \( w := e \).
   We have \( C_2 = C'_2 = \text{stop} \). Hypothesis \( M'_1 \subseteq_e M_1 \) implies \( M_1(e) = M'_1(e) \), and thus, \( M_2(w) = M'_2(w) \). Because \( M'_2(T(w)) = M'_1(T(e)) \cup M'_1([cc]) \cup M'_1(bc) \) and \( M_2(T(w)) = M_1(T(e)) \cup M_1([cc]) \cup M_1(bc) \), hypothesis \( M'_1 \subseteq_e M_1 \) gives \( M'_2(T(w)) \subseteq M_2(T(w)) \). So, \( M'_2 \subseteq_e M_2 \).

3. \( C_1 \) is \( \text{exit} \).
   We have \( C_2 = C'_2 = \text{stop} \). Because \( M_2(cc) = M_1(cc).\text{pop} \) and \( M'_2(cc) = M'_1(cc).\text{pop} \), hypothesis \( M'_1 \subseteq_e M_1 \) gives \( \forall i : M'_2([cc.\text{pop}^i]) \subseteq M_2([cc.\text{pop}^i]) \).

Thus, \( E_{H,L} \) produces observation \( \langle l, 1 \rangle \), but 2-\textit{Enf} does not. So, \( E_{H,L} \not<_{c,L} 2-\text{Enf} \). From 2-\textit{Enf} \( \not<_{c,L} E_{H,L} \), we then have 2-\textit{Enf} \( \not<_{c,L} E_{H,L} \). Notice that the same program \( \text{pgm} \) works for 2-\textit{Eopt}, too. And thus we get 2-\textit{Eopt} \( \not<_{c,L} E_{H,L} \). \( \square \)
Hypothesis $M'_1 \sqsubseteq_e M_1$ also gives $M'_1(\text{cc}) \sqsubseteq M_1(\text{cc})$, $M'_1(\text{cc.top.A}) = M_1(\text{cc.top.A})$, and $M'_1(\text{bc}) \sqsubseteq M_1(\text{bc})$, and thus, we get $M'_2(\text{bc}) \sqsubseteq M_2(\text{bc})$. From $M'_1 \sqsubseteq_e M_1$, we also get $M'_1(\text{cc.top.W}) = M_1(\text{cc.top.W})$. Because, for all $w \in W$, we have $M'_2(T(w)) = M'_1(T(w)) \sqcup M'_1(\lfloor \text{cc} \rfloor) \sqcup M'_1(\text{bc})$ and $M_2(T(w)) = M_1(T(w)) \sqcup M_1(\lfloor \text{cc} \rfloor) \sqcup M_1(\text{bc})$, 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-\text{Enf} and $E^{H,L}$ use the same rules for other commands, so $M'_2 \sqsubseteq_e M_2$ follows easily.

\hfill \Box

\textbf{Lemma 24.} $E^{H,L}$ is an enforcer and uses 1-dependent $G_{a:=e}$.

\textbf{Proof.} It is easy to prove that $E^{H,L}$ 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 $E^{H,L}$ 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 (\textsc{AsgnA}) of $E^{H,L}$ we get that $G_{a:=e}$ is $M(T(e)) \sqcup M(\lfloor \text{cc} \rfloor) \sqcup M(\text{bc}) \sqsubseteq M(T(a))$. Because $a := e$ is in an anchor-tailed command, we get that $a := e$ is not encapsulated in any conditional command. So, $M(\text{cc}) = \epsilon$. Because $C$ does not involve any assignment to anchor variable, $bc$ is $\bot$ when execution reaches $C'$. While executing $C'$, $cc$ remains $\epsilon$, and thus, from rule (\textsc{AsgnA}) of $E^{H,L}$, we get that $bc$ remains $\bot$. So, $M(\text{bc}) = \bot$. Thus, $G_{a:=e}$ is $M(T(e)) \sqsubseteq M(T(a))$. So, $E^{H,L}$ uses 1-dependent $G_{a:=e}$.

\hfill \Box

\textbf{References}


