Chapter 9

Information Flow Control

Information flow policies specify whether or not the values in given sets of variables are allowed to affect the values in other sets of variables.[1] If an adversary somehow can observe that a program's execution will not terminate, then an information flow policy also might specify what set of variables are allowed to affect program termination. Many confidentiality and integrity policies have natural formulations as information flow policies. An information flow policy that prohibits values in the set of secret variables from affecting values in the set of public variables is specifying confidentiality; integrity would be analogous, but with values read from the set of untrusted variables prohibited from affecting the set of trusted variables.

With *static enforcement*, restrictions are packaged as a type system, and the type-correctness of a program is checked by a compiler prior to executing that program; with *dynamic enforcement*, restrictions are checked at runtime by a reference monitor. Elements of static enforcement and dynamic enforcement are combined in *hybrid enforcement*.

Information flow policies are *end-to-end*—they restrict uses of system inputs, including uses of values derived from those inputs. The other authorization policies discussed in this text have not restricted uses of values, so we must trust that programs do not abuse values they read. A policy that is not end-to-end, for example, cannot prohibit public outputs from being affected by secret inputs during execution of a program that is authorized to read those secrets from input channels and authorized to write to public output channels.

[1] Since input channels and output channels can be represented by sequence-valued variables, using sets of variables is not really a restriction.

9.1 Information Flow Policies

An information flow policy (i) associates each variable v with a label $\Gamma(v)$ from a finite set Λ, and (ii) defines a partial order[2] $\sqsubseteq$ (and its complement $\not\sqsubseteq$) on those labels. Variables having the same label λ form sets V_λ, and partial order $\sqsubseteq$ specifies whether one set of variables is allowed to affect another, as follows.

- $\lambda' \sqsubseteq \lambda$ specifies that variables in $V_{\lambda'}$ are allowed to affect variables in V_λ.
- $\lambda' \not\sqsubseteq \lambda$ specifies that variables in $V_{\lambda'}$ are not allowed to affect variables in V_λ.

Taking into account the transitivity of $\sqsubseteq$, we get the following characterization of what is required for compliance with an information flow policy.

> **Prohibited Information Flows.** For all labels $\lambda \in \Lambda$, variables in set $V_{\not\sqsubseteq\lambda}$ are not allowed to affect variables in set $V_{\sqsubseteq\lambda}$, where $V_{\not\sqsubseteq\lambda}$ and $V_{\sqsubseteq\lambda}$ are defined by:
>
> $$V_{\not\sqsubseteq\lambda} = \bigcup_{\iota \not\sqsubseteq \lambda} V_\iota \qquad\qquad V_{\sqsubseteq\lambda} = \bigcup_{\iota \sqsubseteq \lambda} V_\iota \qquad\qquad \square$$

To see why Prohibited Information Flows is implied by the restrictions partial order $\sqsubseteq$ specifies about whether one variable can affect another, we show that these restrictions imply variables $v \in V_{\not\sqsubseteq\lambda}$ are prohibited from affecting variables $w \in V_{\sqsubseteq\lambda}$. For $v \in V_{\not\sqsubseteq\lambda}$ to hold, there must be a label λ' such that $\lambda' \not\sqsubseteq \lambda$ and $v \in V_{\lambda'}$ hold. Similarly, for $w \in V_{\sqsubseteq\lambda}$ to hold, there must exist a label λ'' such that $\lambda'' \sqsubseteq \lambda$ and $w \in V_{\lambda''}$ hold. Variable v is prohibited by $\sqsubseteq$ from affecting variable w if $\lambda' \not\sqsubseteq \lambda''$ holds. So we prove that $\lambda' \not\sqsubseteq \lambda''$ holds. The proof is by contradiction—we assume that $\lambda' \sqsubseteq \lambda''$ holds and prove this assumption implies *false* given conditions $\lambda' \not\sqsubseteq \lambda$ and $\lambda'' \sqsubseteq \lambda$ that we know hold. From assumption $\lambda' \sqsubseteq \lambda''$, transitivity of $\sqsubseteq$ with $\lambda'' \sqsubseteq \lambda$ derives $\lambda' \sqsubseteq \lambda$. But $\lambda' \sqsubseteq \lambda$ and $\lambda' \not\sqsubseteq \lambda$ together imply *false*, completing the proof.

Information Flow Policy Example. An obvious application of Prohibited Information Flows is for defining a confidentiality policy that prohibits leaks of secrets. We use $\{\mathsf{L}, \mathsf{H}\}$ for Λ. A variable storing secret information is given label H (for High); a variable storing public information is given label L (for Low). To specify that secrets should not be leaked, we give relation $\sqsubseteq$ and its complement

[2] A *relation* ρ on a set *Vals* is a set of pairs $\{\langle a, b\rangle \mid a, b \in \mathit{Vals}\}$ and its *complement* $\not\rho$ is the set of pairs $\{\langle a, b\rangle \mid a, b \in \mathit{Vals} \wedge \langle a, b\rangle \notin \rho\}$. By convention, infix notation $a \,\rho\, b$ is used to indicate that $\langle a, b\rangle \in \rho$ holds. A relation ρ on a set *Vals* is defined to be a *partial order* if it satisfies the following.

> Reflexive: $a \,\rho\, a$ for all $a \in \mathit{Vals}$.
> Antisymmetric: $a \,\rho\, b$ and $b \,\rho\, a$ implies $a = b$ for all $a, b \in \mathit{Vals}$.
> Transitive: $a \,\rho\, b$ and $b \,\rho\, c$ implies $a \,\rho\, c$ for all $a, b, c \in \mathit{Vals}$.

A partial order ρ does not have to relate all pairs of elements $a, b \in \mathit{Vals}$, so if $a \not\rho\ b$ holds it is possible that neither $a \,\rho\, b$ nor $b \,\rho\, a$ holds.

$\sqsubseteq$	L	H
L	$\sqsubseteq$	$\sqsubseteq$
H	$\not\sqsubseteq$	$\sqsubseteq$

(a) Definition of $\sqsubseteq$

$\sqcup$	L	H
L	L	H
H	H	H

(b) Definition of $\sqcup$

Figure 9.1: Definitions for $\Lambda = \{\mathsf{H}, \mathsf{L}\}$

$\not\sqsubseteq$ using the table of Figure 9.1(a): $\mathsf{L} \sqsubseteq \mathsf{L}$, $\mathsf{L} \sqsubseteq \mathsf{H}$, $\mathsf{H} \not\sqsubseteq \mathsf{L}$, and $\mathsf{H} \sqsubseteq \mathsf{H}$. Because $\mathsf{H} \not\sqsubseteq \mathsf{L}$ is specified, variables with label H are not allowed to affect variables with label L. So the information flow policy prohibits the values in secret variables from affecting the values in public variables, as desired.

This same information flow policy also works for specifying an integrity policy that asserts variables storing untrusted values are not allowed to contaminate values in variables intended to store trusted values. Notice, the prohibitions on untrusted values in this integrity policy are analogous to secret values in the above confidentiality policy. So the prohibitions required by the integrity policy follow from $\mathsf{H} \not\sqsubseteq \mathsf{L}$ specified in the table of Figure 9.1(a) if variables storing untrusted values are given label H and variables storing trusted values are given label L.

For a more conversational characterization of the information flow policies just described, think of an information flow from a variable with label L to a variable with label H as information moving "up", and think of an information flow from label H to label L as information moving "down". We consider a single read or write action performed by a variable x to a variable y, and we define $\lambda \sqsubset \lambda'$ to be an abbreviation for $\lambda \sqsubseteq \lambda' \wedge \lambda \neq \lambda'$. We have:

action by x	$\sqsubseteq$ relation	description
x reads y	$\Gamma(x) \sqsubset \Gamma(y)$	x "reads up"
	$\Gamma(y) \sqsubset \Gamma(x)$	x "reads down"
x writes y	$\Gamma(x) \sqsubset \Gamma(y)$	x "writes up"
	$\Gamma(y) \sqsubset \Gamma(x)$	x "writes down"

The information flow policy of Figure 9.1 specifies a single prohibition: $\mathsf{H} \not\sqsubseteq \mathsf{L}$, meaning a variable with label H is not allowed to affect a variable with label L. In terms of read and write actions, that prohibition means

- a variable with label L is not allowed to read from a variable with label H: "no read up"
- a variable with label H is not allowed to write to a variable with label L: "no write down"

So the information flow policy of Figure 9.1 specifies: "no read up; no write down".

When "no read up; no write down" is applied to confidentiality, "no read up" says that public variables (label is L) cannot read from secret variables (label is H); "no write down" says that secret variables (label is H) cannot be written into public variables (label is L). When applied to integrity, "no read up" says that trusted variables (label is L) cannot read from untrusted variables (label is H); "no write down" says that untrusted variables (label is H) cannot be written into trusted variables (label is L).

Why $\sqsubseteq$ is a partial order. A specious distinction is being made in any information flow policy that employs different labels λ' and λ'' but $\lambda' \sqsubseteq \lambda''$ and $\lambda'' \sqsubseteq \lambda'$ both hold. We avoid such specious distinctions by requiring that if $\Gamma(v) \neq \Gamma(w)$ holds for variables v and w then the information flow policy specifies that (i) variable v is allowed to affect different variables than variable w is allowed to affect or (ii) variable v is allowed to be affected by different variables than may affect w. So we prohibit information flow policies from having pairs of labels λ' and λ'' that satisfy

$$(\forall \lambda \in \Lambda\colon\ \ (\lambda' \sqsubseteq \lambda) = (\lambda'' \sqsubseteq \lambda)\ \ \wedge\ \ (\lambda \sqsubseteq \lambda') = (\lambda \sqsubseteq \lambda'')),$$

since $\sqsubseteq$ then would be creating specious distinctions between variables with label λ' and those with label λ''. Furthermore, if a variable v is allowed to affect a variable w and also w is allowed to affect v then we require that an information flow policy assign the same label to both v and w.

If these requirements on when labels can be different and when they must be the same are satisfied then relation $\sqsubseteq$ will satisfy two properties.

- $\sqsubseteq$ is reflexive: $\lambda \sqsubseteq \lambda$ for all $\lambda \in \Lambda$. This follows if a variable v is allowed to affect any other variable w having the same label, since if $\Gamma(v) = \Gamma(w)$ holds then $\Gamma(v) \sqsubseteq \Gamma(w)$ will hold.

- $\sqsubseteq$ is antisymmetric: $(\lambda \sqsubseteq \lambda' \ \wedge\ \lambda' \sqsubseteq \lambda) \Rightarrow \lambda = \lambda'$. This follows if variables v and w must have the same label when v is both allowed to affect w ($\Gamma(v) \sqsubseteq \Gamma(w)$ holds) and also v is allowed to be affected by the other ($\Gamma(w) \sqsubseteq \Gamma(v)$ holds).

In addition, because "cause and effect" could be transitive, it seems prudent for relation $\sqsubseteq$ to be transitive.

- $\sqsubseteq$ is transitive: $(\lambda \sqsubseteq \lambda' \ \wedge\ \lambda' \sqsubseteq \lambda'') \Rightarrow \lambda \sqsubseteq \lambda''$. If a variable v is allowed to affect a variable w because $\Gamma(v) \sqsubseteq \Gamma(w)$ holds, and if w is allowed to affect a variable x because $\Gamma(w) \sqsubseteq \Gamma(x)$ holds, then $\Gamma(v) \sqsubseteq \Gamma(x)$ should hold so that variable v is allowed to affect variable x.

So there are good reasons for $\sqsubseteq$ to be reflexive, antisymmetric, and transitive. That means we have justification for requiring that $\sqsubseteq$ be a partial order which, by definition (see footnote 2 on page 244), must be reflexive, antisymmetric, and transitive.

Labels for Expressions. By associating a label $\Gamma(E)$ with an expression E, an information flow policy can specify what is allowed to be affected by E and what E is allowed to affect. The information flow policy would specify that $\lambda \sqsubseteq \Gamma(E)$ holds to indicate that a variable or expression with label λ is allowed to affect the value of E; it would specify that $\Gamma(E) \sqsubseteq \lambda$ holds to indicate that the value of E is allowed to affect the value of a variable or expression with label λ.

Constructing labels for expressions is simplified if we ignore the semantics of operators and functions.

> **Variables Affecting Expressions.** The value of an expression E is allowed to be affected by any and all of the variables E mentions. □

This characterization means label $\Gamma(E)$ must satisfy $\Gamma(v) \sqsubseteq \Gamma(E)$ for all variables v mentioned in E.

The variables referenced in an expression might not all have the same label. For an expression E, the set Λ_E of labels on variables mentioned in E might not contain a label λ_E that satisfies $\lambda \sqsubseteq \lambda_E$ for all $\lambda \in \Lambda_E$, as required by Variables Affecting Expressions. So to construct $\Gamma(E)$ we require that every subset Λ' of Λ have a corresponding *least upper bound* that is itself a label in Λ.[3] This least upper bound would be a label that is at least as large as any label in Λ' but not larger than necessary. Formally, for each subset $\Lambda' \subseteq \Lambda$ there would be a label $\lambda_{lub(\Lambda')}$ where, by definition, (i) $\lambda_i \sqsubseteq \lambda_{lub(\Lambda')}$ holds for all $\lambda_i \in \Lambda'$ and (ii) for any label $\hat{\lambda}$ satisfying $\hat{\lambda} \sqsubseteq \lambda_{lub(\Lambda')}$ there would be some $\lambda_i \in \Lambda'$ where $\lambda_i \not\sqsubseteq \lambda_{lub(\Lambda')}$ holds. Property (i) that says $\lambda_{lub(\Lambda')}$ is at least as large as any label in subset Λ'; property (ii) says that $\lambda_{lub(\Lambda')}$ is not larger than necessary.

We can construct a least upper bound by using a commutative and associative *join* operator $\sqcup$ that evaluates to the least upper bound of its arguments and, therefore, satisfies:

$$\lambda \sqcup \lambda \;=\; \lambda \tag{9.1}$$

$$\lambda_i \;\sqsubseteq\; (\lambda_1 \sqcup \lambda_2 \sqcup \cdots \sqcup \lambda_n) \text{ for } 1 \le i \le n \tag{9.2}$$

$$\neg(\exists \hat{\lambda} \in \Lambda\colon\; \hat{\lambda} \sqsubset (\lambda_1 \sqcup \lambda_2 \sqcup \cdots \sqcup \lambda_n) \;\wedge\; (\forall i\colon\; \lambda_i \sqsubseteq \hat{\lambda})) \tag{9.3}$$

[3] A partially ordered set that contains a least upper bound for every finite subset is called a *join-semilattice*. A simple example of a join-semilattice is constructed by having the elements of the join-semilattice be all subsets of a finite set. For subsets P and Q: $P \sqsubseteq Q$ is defined to be $P \subseteq Q$, and least upper bound $P \sqcup Q$ of P and Q is defined to be $P \cup Q$.

A similar construction allows labels on variables to serve as the elements of a join-semilattice. For labels λ and λ', we define $\lambda \sqsubseteq \lambda'$as follows.

$$\lambda \sqsubseteq \lambda'\colon\; V_{\sqsubseteq\lambda} \subseteq V_{\sqsubseteq\lambda'}$$

We define least upper bound $\lambda_{lub(\Lambda')}$ for a subset Λ' of the set Λ of labels to be the smallest label $\lambda' \in \Lambda$ satisfying

$$(\bigcup_{\lambda \in \Lambda'} V_{\sqsubseteq\lambda}) \;\subseteq\; V_{\sqsubseteq\lambda'}$$

As an example, Figure 9.1(b) gives a table that defines the $\sqcup$ operator for the information flow policy we have been discussing that uses labels $\{\mathsf{L}, \mathsf{H}\}$ to prohibit leaks of secrets.

Property (9.2) implies that $\Gamma(v_1) \sqcup \Gamma(v_2) \sqcup \cdots \sqcup \Gamma(v_n)$ can serve as label $\Gamma(E)$ for an expression E that references variables v_1, v_2, ..., v_n, since $\Gamma(v_i) \sqsubseteq \Gamma(E)$ holds for each v_i as required for Variables Affecting Expressions. Property (9.3) rules out choosing for $\Gamma(E)$ a label that is more restrictive than needed for satisfying (9.2). Together, they justify:

> **Construction of $\Gamma(E)$.** The label $\Gamma(E)$ for an expression E written in terms of variables v_1, v_2, ... , v_n is: $\Gamma(v_1) \sqcup \Gamma(v_2) \sqcup \cdots \sqcup \Gamma(v_n)$. □

So, for example, the label $\Gamma(x + y)$ for expression $x + y$ would be $\Gamma(x) \sqcup \Gamma(y)$. Due to (9.2), this label satisfies $\Gamma(x) \sqsubseteq \Gamma(x) \sqcup \Gamma(y)$ and $\Gamma(y) \sqsubseteq \Gamma(x) \sqcup \Gamma(y)$. That means this label specifies that x and y each is allowed to affect $x+y$, as Variables Affecting Expressions prescribes. It also means that if x and y are assigned labels from $\{\mathsf{H}, \mathsf{L}\}$ then, according to Figure 9.1(b), $\Gamma(x+y) = \mathsf{L}$ holds if $\Gamma(x) = \Gamma(y) = \mathsf{L}$ holds, but $\Gamma(x + y) = \mathsf{H}$ holds otherwise. This label assignment for expression $x+y$ should not be surprising—when secret information is combined with public information the value that results could reveal secrets, so it is appropriate to label that value with H.

Our label construction for expressions, however, also derives $\Gamma(x) \sqcup \Gamma(y)$ as the label for expression $x + y - y$. Expression $x + y - y$ is not influenced by y, which means our label construction produced an unnecessarily restrictive label. The label is unnecessarily restrictive because label $\Gamma(x) \sqcup \Gamma(y)$ prohibits values of $x + y - y$ from being used where values are not allowed to be influenced by y—even though the value of $x+y-y$ is not influenced by y. We conclude that our label construction for expressions is conservative. Less conservative schemes are possible by using the semantics of the operators and functions; the complexity of these schemes can be formidable, though.

9.1.1 Termination Insensitive Noninterference (TINI)

To characterize whether a deterministic program or statement S complies with an information flow policy, we define a relation[4] $V \overset{S}{\nmapsto}_{\mathsf{ti}} W$ that is satisfied if and only if terminating executions of S that start in states differing only in the values for the variables in V terminate in states where corresponding variables in W have the same value. Thus, if $V_{\lambda'} \overset{S}{\nmapsto}_{\mathsf{ti}} V_{\lambda}$ holds for all statements S of a program then that program complies with the restrictions that $\lambda' \sqsubseteq \lambda$ specifies, because the values in variables $V_{\lambda'}$ do not affect the values in V_{λ}.

To give a formal defintion for relation $V \overset{S}{\nmapsto}_{\mathsf{ti}} W$, we describe executions of

[4] Subscript ti on $\overset{S}{\nmapsto}_{\mathsf{ti}}$ abbreviates termination insensitive and conveys that nonterminating executions are being ignored.

S by using a function from initial states s:

$$[\![S]\!](s)\colon \begin{cases} s' & \text{if executing } S \text{ in state } s \text{ terminates in state } s' \\ \bot & \text{if executing } S \text{ in state } s \text{ is nonterminating} \end{cases} \tag{9.4}$$

We also employ some predicates on states. In these, $Vars(S)$ denotes the set of variables in S, $\overline{V}$ denotes $Vars(S) - V$ for $V \subseteq Vars(S)$, and $s.v$ denotes the value of a variable $v \in Vars(S)$ in state s:

$s =_V s'$: $\quad s.v = s'.v$ for all $v \in V$

$s \neq_V s'$: $\quad s.v \neq s'.v$ for some $v \in V$

$s =_{\overline{V}} s'$: $\quad s.v = s'.v$ for all $v \in Vars(S) - V$

$s \neq_{\overline{V}} s'$: $\quad s.v \neq s'.v$ for some $v \in Vars(S) - V$

Relation $V \overset{S}{\nrightarrow}_{\mathsf{ti}} W$ is then formally defined by the following.

$$(\forall s, s'\colon\ s =_{\overline{V}} s' \ \wedge\ [\![S]\!](s) \neq \bot \ \wedge\ [\![S]\!](s') \neq \bot \ \Rightarrow\ [\![S]\!](s) =_W [\![S]\!](s')) \tag{9.5}$$

This formula is an assertion about all pairs of terminating executions from initial states s and s' satisfying $s =_{\overline{V}} s'$—initial states that may differ only in the values of variables in V (since $s =_{\overline{V}} s'$ requires that the values of all other variables are equal). The formula asserts that executions of S from these (possibly) different initial states produce final states $[\![S]\!](s)$ and $[\![S]\!](s')$ satisfying $[\![S]\!](s) =_W [\![S]\!](s')$. So these final states agree on the values for variables in W, which establishes that the values of variables in W are unaffected by the differences in the initial states s and s'.

The following rule allows $\overset{S}{\nrightarrow}_{\mathsf{ti}}$ relations be combined. It generalizes two observations: (i) if variables in neither V or V' are allowed to affect variables in W then variables in $V \cup V'$ are not allowed to affect variables in W, and (ii) if variables in V are not allowed to affect variables in W or in W' then variables in V are not allowed to affect variables in $W \cup W'$.

$$(\forall i \in I, j \in J\colon\ V_i \overset{S}{\nrightarrow}_{\mathsf{ti}} V_j) \ \Rightarrow\ (\bigcup_{i \in I} V_i) \overset{S}{\nrightarrow}_{\mathsf{ti}} (\bigcup_{j \in J} V_j)$$

Using this rule in conjunction with the definitions for $V_{\not\in\lambda}$ and $V_{\sqsubseteq\lambda}$ in Prohibited Information Flows (page 244), we derive for later use that for all $\lambda \in \Lambda$:

$$(\forall \lambda' \not\in \lambda, \lambda'' \sqsubseteq \lambda\colon\ V_{\lambda'} \overset{S}{\nrightarrow}_{\mathsf{ti}} V_{\lambda''}) \ \Rightarrow\ V_{\not\in\lambda} \overset{S}{\nrightarrow}_{\mathsf{ti}} V_{\sqsubseteq\lambda} \tag{9.6}$$

The complement of $V \overset{S}{\nrightarrow}_{\mathsf{ti}} W$ is a relation $V \overset{S}{\longmapsto}_{\mathsf{ti}} W$ defined by the negation of formal definition (9.5) for $V \overset{S}{\nrightarrow}_{\mathsf{ti}} W$.[5]

$$(\exists s, s'\colon\ s =_{\overline{V}} s' \ \wedge\ [\![S]\!](s) \neq \bot \ \wedge\ [\![S]\!](s') \neq \bot \ \wedge [\![S]\!](s) \neq_W [\![S]\!](s')) \tag{9.7}$$

[5]Recall from predicate logic: $\neg(\forall x\colon\ P) = (\exists x\colon\ \neg P)$ and $\neg(P \Rightarrow Q) = \neg(\neg P \vee Q) = P \wedge \neg Q$.

$V \overset{S}{\longmapsto}_{\mathsf{ti}} W$ asserts the existence of a pair of terminating executions from initial states s and s' satisfying $s =_{\overline{V}} s'$—that is, a pair of executions from initial states that may differ only in the values of variables in V. For (9.7) to hold, these executions must produce final states $[\![S]\!](s)$ and $[\![S]\!](s')$ satisfying $[\![S]\!](s) \neq_W [\![S]\!](s')$. So the final states must differ in the values for some variables in W, establishing that some differences in the initial values of variables of V did lead to differences in the final values of variables in W.

In programs that comply with an information flow policy, there is a connection between $\overset{S}{\not\longmapsto}_{\mathsf{ti}}$ and $\sqsubseteq$. This connection can be formulated in two ways, one the contrapositive of the other.[6] The first asserts prohibited information flows that $\not\sqsubseteq$ specifies are enforced. The second asserts $\lambda' \sqsubseteq \lambda$ should hold if variables labeled λ' are going to affect variables labeled λ.

> **Label Invariant.** For all statements S' in S:
>
> (a) $\lambda' \not\sqsubseteq \lambda \;\Rightarrow\; V_{\lambda'} \overset{S'}{\not\longmapsto}_{\mathsf{ti}} V_{\lambda}$
>
> (b) $V_{\lambda'} \overset{S'}{\longmapsto}_{\mathsf{ti}} V_{\lambda} \;\Rightarrow\; \lambda' \sqsubseteq \lambda$

Repeated applications of Label Invariant (a) derives for every $\lambda \in \Lambda$

$$(\forall \lambda' \not\sqsubseteq \lambda,\, \lambda'' \sqsubseteq \lambda\colon\; V_{\lambda'} \overset{S}{\not\longmapsto}_{\mathsf{ti}} V_{\lambda''}), \tag{9.8}$$

as follows. First, observe that $\lambda' \not\sqsubseteq \lambda \wedge \lambda'' \sqsubseteq \lambda \Rightarrow \lambda' \not\sqsubseteq \lambda''$ holds. The proof is by contradiction: $\lambda' \not\sqsubseteq \lambda \wedge \lambda'' \sqsubseteq \lambda \wedge \lambda' \sqsubseteq \lambda''$ is equivalent to *false*, since (by transitivity) it implies $\lambda' \not\sqsubseteq \lambda \wedge \lambda' \sqsubseteq \lambda$. Therefore if $\lambda' \not\sqsubseteq \lambda \wedge \lambda'' \sqsubseteq \lambda$ holds, then $\lambda' \not\sqsubseteq \lambda''$ holds, which means, due to Label Invariant (a), that $V_{\lambda'} \overset{S}{\not\longmapsto}_{\mathsf{ti}} V_{\lambda''}$ holds, as required for proving (9.8). Moreover, having proved (9.8), we can conclude from (9.6) that $V_{\not\sqsubseteq\lambda} \overset{S}{\not\longmapsto}_{\mathsf{ti}} V_{\sqsubseteq\lambda}$ holds. Generalizing to any label label $\lambda \in \Lambda$, we get:

> **Termination Insensitive Noninterference (TINI).** If a deterministic program S complies with an information flow policy defined by a set Λ of labels with a partial order $\sqsubseteq$ then executions of S will satisfy: $(\forall \lambda \in \Lambda\colon\; V_{\not\sqsubseteq\lambda} \overset{S}{\not\longmapsto}_{\mathsf{ti}} V_{\sqsubseteq\lambda})$. □

As an illustration of TINI, we return to the information flow policy specified by $\Lambda = \{\mathsf{L}, \mathsf{H}\}$, where $\mathsf{L} \sqsubseteq \mathsf{H}$ holds. TINI for a deterministic program S thus specifies the following.

$$V_{\not\sqsubseteq\mathsf{L}} \overset{S}{\not\longmapsto}_{\mathsf{ti}} V_{\sqsubseteq\mathsf{L}} \quad\wedge\quad V_{\not\sqsubseteq\mathsf{H}} \overset{S}{\not\longmapsto}_{\mathsf{ti}} V_{\sqsubseteq\mathsf{H}} \tag{9.9}$$

[6] The contrapositive of predicate logic formula $P \Rightarrow Q$ is $\neg Q \Rightarrow \neg P$. Each is equivalent to $\neg P \vee Q$, so contrapositives are equivalent.

$\Gamma(in)$	$\Gamma(out)$	$V_{\sqsubseteq \mathsf{L}}$	$V_{\not\sqsubseteq \mathsf{L}}$	$V_{\sqsubseteq \mathsf{H}}$	$V_{\not\sqsubseteq \mathsf{H}}$	$out := in?$
L	L	$\{in, out\}$	$\varnothing$	$\{in, out\}$	$\varnothing$	$\surd$
L	H	$\{in\}$	$\{out\}$	$\{in, out\}$	$\{\varnothing\}$	$\surd$
H	L	$\{out\}$	$\{in\}$	$\{in, out\}$	$\{\varnothing\}$	$\times$
H	H	$\varnothing$	$\{in, out\}$	$\{in, out\}$	$\varnothing$	$\surd$

Figure 9.2: Possible Information Flow Policies for $out := in$

By using formal definition (9.5) to expand $\overset{S}{\mapsto\!\!\!\!\!/\!\!\!\!\!\rightarrow}_{\mathsf{ti}}$, we obtain the restrictions on initial and final states of S that compliance with TINI requires:

$$\begin{aligned}
&(\forall s, s' :\ s =_{\overline{V}_{\not\sqsubseteq \mathsf{L}}} s' \ \wedge\ [\![S]\!](s) \neq \bot \ \wedge\ [\![S]\!](s') \neq \bot \\
&\qquad \Rightarrow\ [\![S]\!](s) =_{V_{\sqsubseteq \mathsf{L}}} [\![S]\!](s')) \\
\wedge\ &(\forall s, s' :\ s =_{\overline{V}_{\not\sqsubseteq \mathsf{H}}} s' \ \wedge\ [\![S]\!](s) \neq \bot \ \wedge\ ;[\![S]\!](s') \neq \bot \\
&\qquad \Rightarrow\ [\![S]\!](s) =_{V_{\sqsubseteq \mathsf{H}}} [\![S]\!](s'))
\end{aligned}$$

Simplification of this formula is possible, because the following hold

$$\overline{V}_{\not\sqsubseteq \mathsf{L}} = V_{\mathsf{L}} \qquad V_{\sqsubseteq \mathsf{L}} = V_{\mathsf{L}} \qquad \overline{V}_{\not\sqsubseteq \mathsf{H}} = V_{\sqsubseteq \mathsf{H}} \qquad V_{\sqsubseteq \mathsf{H}} = V_{\mathsf{L}} \cup V_{\mathsf{H}}$$

resulting in

$$\begin{aligned}
&(\forall s, s' :\ s =_{V_{\mathsf{L}}} s' \ \wedge\ [\![S]\!](s) \neq \bot \ \wedge\ [\![S]\!](s') \neq \bot \\
&\qquad \Rightarrow\ [\![S]\!](s) =_{V_{\mathsf{L}}} [\![S]\!](s')) \\
\wedge\ &(\forall s, s' :\ s =_{V_{\mathsf{L}} \cup V_{\mathsf{H}}} s' \ \wedge\ [\![S]\!](s) \neq \bot \ \wedge\ [\![S]\!](s') \neq \bot \\
&\qquad \Rightarrow\ [\![S]\!](s) =_{V_{\mathsf{L}} \cup V_{\mathsf{H}}} [\![S]\!](s'))
\end{aligned} \tag{9.10}$$

Since Λ is $\{\mathsf{L}, \mathsf{H}\}$, we have that $V_{\mathsf{L}} \cup V_{\mathsf{H}} = \mathit{Vars}(S)$ holds, which implies that $s =_{V_{\mathsf{L}} \cup V_{\mathsf{H}}} s'$ is equivalent to asserting $s = s'$ holds. That assertion means the second quantified formula of (9.10) is satisfied due to the assumption that S is deterministic, since terminating executions of S that start from the same initial states then will produce the same final states. Thus, that second part of (9.10) is equivalent to *true*, and we conclude that enforcing TINI for the case where $\Lambda = \{\mathsf{L}, \mathsf{H}\}$ and $\mathsf{L} \sqsubseteq \mathsf{H}$ holds is equivalent to:

$$(\forall s, s' :\ s =_{V_{\mathsf{L}}} s' \ \wedge\ [\![S]\!](s) \neq \bot \ \wedge\ [\![S]\!](s') \neq \bot \ \Rightarrow\ [\![S]\!](s) =_{V_{\mathsf{L}}} [\![S]\!](s'))$$

Since states satisfying $s =_{V_{\mathsf{L}}} s'$ differ only in the values of variables in V_{H} and agree on the values of variables in V_{L}, TINI is specifying that variables in V_{H} have no affect on variables in V_{L}—the values of variables with label H are prohibited from affecting the values of variables with label L. So if TINI is enforced we can use variables with label H to store "secret" information that we do not want leaked to variables with label L.

As a concrete application, consider a program comprising the single assignment statement $out := in$. It is deterministic, it always terminates, and its execution causes the value of variable in to affect the value of variable out. Figure 9.2 summarizes compliance with TINI for all possible labelings of variables in and out. The one row with an × in the final column corresponds to the sole labeling ($\Gamma(in) = \mathsf{H}$ and $\Gamma(out) = \mathsf{L}$) where compliance with TINI prohibits execution of $out := in$. That prohibition should not be surprising—it prohibits executions where a variable having label H affects a variable having label L. Formally, when $\Gamma(in) = \mathsf{H}$ and $\Gamma(out) = \mathsf{L}$ hold, instantiating TINI specification (9.9) for $out := in$ gets:

$$V_{\not\in \mathsf{L}} \overset{out := in}{\mapsto\!\!\!\!\!\!\rightarrow}_{\mathsf{ti}} V_{\in \mathsf{L}} \quad \wedge \quad V_{\not\in \mathsf{H}} \overset{out := in}{\mapsto\!\!\!\!\!\!\rightarrow}_{\mathsf{ti}} V_{\in \mathsf{H}}$$

$$= \text{substitution for } V_{\not\in \mathsf{L}}, V_{\in \mathsf{L}}, V_{\not\in \mathsf{H}}, V_{\in \mathsf{H}} \text{ according to Figure 9.2}$$

$$\{in\} \overset{out := in}{\mapsto\!\!\!\!\!\!\rightarrow}_{\mathsf{ti}} \{out\} \quad \wedge \quad \varnothing \overset{out := in}{\mapsto\!\!\!\!\!\!\rightarrow}_{\mathsf{ti}} \{in, out\}$$

$$= \text{expand } \overset{S}{\mapsto\!\!\!\!\!\!\rightarrow}_{\mathsf{ti}} \text{ according to formalization (9.5)}$$

$$false \quad \wedge \quad true$$

The first conjunct equals *false*, because executing $out := in$ in initial states having equal values for the variables in $\overline{\{in\}}$ means executing $out := in$ in initial states where out is the same but in varies. Different values will be stored into out by those executions, as required for the first conjunct. The second conjunct, however, equals *true* since initial states must be the same, so execution of $out := in$ produces final states that have the same value for out.

Some possibly surprising aspects of TINI are illustrated by other programs that use the information flow policy defined earlier: $\Lambda = \{\mathsf{L}, \mathsf{H}\}$ and $\mathsf{H} \not\in \mathsf{L}$ hold. Two variables x_{L} and x_{H} with $\Gamma(x_{\mathsf{L}}) = \mathsf{L}$ and $\Gamma(x_{\mathsf{H}}) = \mathsf{H}$ suffice. This first program shows that TINI might be violated in programs where assignment statements only change variables labeled L.

if $x_{\mathsf{H}} = 0$ **then** $x_{\mathsf{L}} := 1$ **else** $x_{\mathsf{L}} := 2$

However, assignment statements to variables labeled L appearing in the scope of a guard G labeled H does not guarantee that TINI will be violated. The following program illustrates that possibility.

if $x_{\mathsf{H}} = 0$ **then** $x_{\mathsf{L}} := 1$ **else** $x_{\mathsf{L}} := 1$

A second example illustrates that TINI is not always violated in programs where assignment statements store into variables labeled L from variables labeled H. All executions of the following program are nonterminating, and TINI imposes no restrictions on nonterminating executions.

while *true* **do** $x_{\mathsf{L}} := x_{\mathsf{H}}$ **end**

9.1.2 Termination Sensitive Noninterference

In theory, it is impossible to observe that a program's execution will be nonterminating—we can only observe that an execution has not yet terminated. In practice, however, if we measure the execution time consumed and compare that value with a prediction derived from knowledge of the code and inputs then we might be able to deduce that the execution will never terminate.

Deducing that an execution is nonterminating can convey information about the initial values of some set of variables. For example, deducing nontermination for an execution of

while $x = 0$ **do skip end**

reveals that $x = 0$ holds in the initial state, because the loop terminates immediately if the initial state satisfies $x \neq 0$ and the loop is nonterminating in other initial states.

TINI only concerns terminating executions. So compliance with TINI does not prevent attackers from learning about the values of variables by deducing that an execution is nonterminating. That is a reason to consider information flow policies that extend TINI and specify the set V of variables that is allowed to affect nontermination.

Define relation[7] $V \overset{S}{\not\longmapsto}_{\mathsf{ts}} \bot$ to hold if and only if differences in the initial values of variables from set V never affect whether or not the resulting execution of S is nonterminating. To formalize this definition, recall from (9.4) that $[\![S]\!](s) = \bot$ holds if execution of S from initial state s is nonterminating. Therefore, predicate

$$s =_{S\downarrow} s' \colon \quad ([\![S]\!](s) = \bot) = ([\![S]\!](s') = \bot)$$

is satisfied by initial states s and s' if both cause nonterminating executions or both cause terminating executions. The formal definitions for $V \overset{S}{\not\longmapsto}_{\mathsf{ts}} \bot$ is:

$$(\forall s, s' \colon \quad s =_{\overline{V}} s' \;\Rightarrow\; s =_{S\downarrow} s')$$

TINI uses labels to restrict which sets of variables are allowed to affect other sets of variables. We extend TINI to specify sets of variables that are allowed to affect nontermination by defining a label $\Gamma(\bot)$: $\lambda \sqsubseteq \Gamma(\bot)$ holds if and only if variables in V_λ are allowed to affect whether or not the program terminates. So variables in $V_{\sqsubseteq\Gamma(\bot)}$ are allowed to affect whether or not the program terminates, and variables in $\overline{V_{\sqsubseteq\Gamma(\bot)}}$ (equivalently $V_{\not\sqsubseteq\Gamma(\bot)}$) are not allowed to affect whether the program terminates or not. The resulting information flow policy prohibition for nontermination is:

$$V_{\not\sqsubseteq\Gamma(\bot)} \overset{S}{\not\longmapsto}_{\mathsf{ts}} \bot$$

We can now TINI to prohibit the values in variables of $V_{\sqsubseteq\Gamma(\bot)}$ from affecting nontermination as follows.

[7] Subscript ts abbreviates termination sensitive.

Termination Sensitive Information Flow (TSNI). If a deterministic program S complies with an information flow policy defined by a set Λ of labels with a partial order $\sqsubseteq$ then executions of S will satisfy:
$(\forall \lambda \in \Lambda\colon\ V_{\not\sqsubseteq \lambda} \overset{S}{\mapsto\!\!\!\!\!\!\rightarrow}_{\mathsf{ti}} V_{\sqsubseteq \lambda}) \quad \wedge \quad V_{\not\sqsubseteq \Gamma(\bot)} \overset{S}{\mapsto\!\!\!\!\!\!\rightarrow}_{\mathsf{ts}} \bot$ □

Typically $\Gamma(\bot)$ is a label that dominates no other, so $\Gamma(\bot) \sqsubseteq \lambda$ holds for all $\lambda \in \Lambda$, representing the view that nontermination is visible to an observer if any variable is visible to that observer.