Information Flow

Should you press "send error report" ?

Some Concepts

Who may access data?

How may data be used?

user control

- Discretionary Access Control

management control

- Mandatory Access Control

- Decentralised Label Model (Myers)

- Information Flow (Denning, Volpano)

concrete access operations

abstract security operations

Discretionary Access Control

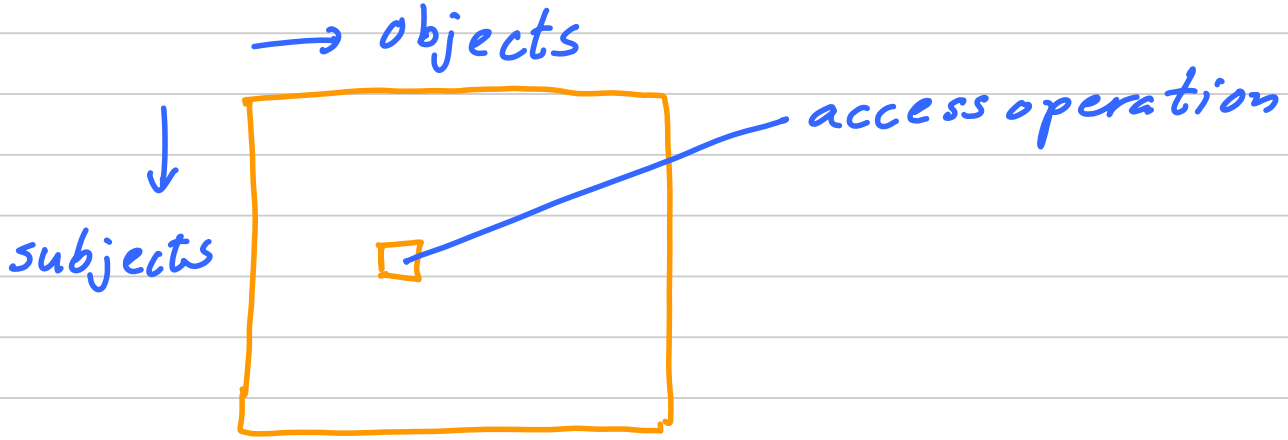
Subjects or principals denote users, programs

Objects denote files, data, ressources

Access denotes operations like read, write,
execute, append, ..., change owner, ...

E.g. UNIX: file: rwxrwxrwx
owner group other

Access Control Matrix



Access Control list: a column

at each object: for (group of) subjects the list of accesses

Capability List : a row

at each subject: for (group of) objects the list of accesses

Mandatory Access Control

Discretionary Access Control is flexible but it is hard to get the matrix right. So we add:

Add security classification
- for each subject
- for each object

Impose security policy for each access

Bell LaPadula: "No read up No write down"

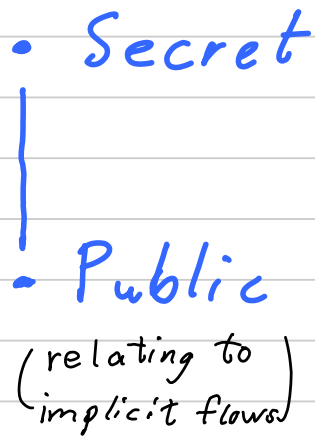
if read then

level (subject) ≥ level (object)

if write then

level (object) ≥ level (subject)

plus some more "technical" conditions (relating to implicit flows)



this should ensure that secret data never ends up in public files

Information Flow

Information Flow applies the security classification of mandatory access control at a lower level of granularity so that a subject can operate on variables at many levels.

The focus is on

- explicit flows: where can sensitive data move?
- implicit flows: under what conditions can data move?

Why do we need to use program analysis techniques like flow analysis and type systems?

Because the obvious alternative of using an execution monitor (a generalised reference monitor) is too weak!

Example:

l := 1; if h = 0 then l := 0 else skip fi

if you block here you leak that h = 0

if you block here you would also need to block

if h = 0 then skip else skip

and this is too restrictive

If you do not block you leak whether or not h = 0

Lattices and Partial Orders

Given a set S and a relation $\sqsubseteq$ on S ($\sqsubseteq : S \times S \to \{tt, ff\}$) we say that

$\sqsubseteq$ is a **preorder** when it is reflexive ($x \sqsubseteq x$) and transitive ($x \sqsubseteq y \wedge y \sqsubseteq z \Rightarrow x \sqsubseteq z$)

$\sqsubseteq$ is a **partial order** when it is a preorder and antisymmetric ($x \sqsubseteq y \wedge y \sqsubseteq x \Rightarrow x = y$)

$\sqsubseteq$ is a **total order** when it is a partial order and total ($x \sqsubseteq y \vee y \sqsubseteq x$ always holds)

Examples

$(\mathbb{Z}, \leq)$ is a [redacted]

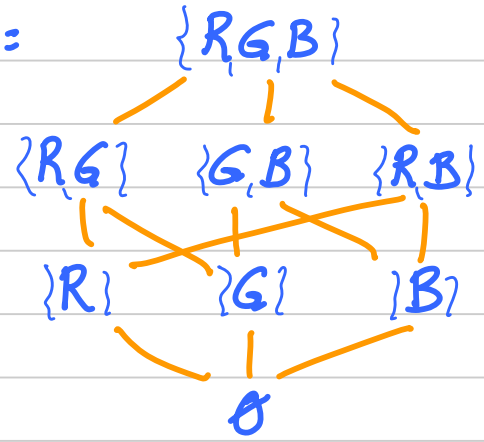
$\mathbb{Z} = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$
$\leq$ as usual

$(BYTE, \leq)$ is a [redacted]

$BYTE = \{0, \ldots, 255\}$
$\leq$ as usual

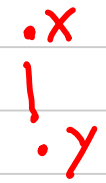
$(\mathcal{P}\{R, G, B\}, \subseteq)$ is a [redacted]

$\mathcal{P}\{R, G, B\} =$



$\subseteq$ is set inclusion

We draw preorders such that



means $y \sqsubseteq x$

but omit reflexive & transitive consequences

Examples (continued)

$(LIST[R,G,B], \sqsubseteq)$ is a [redacted]

$LIST[R,G,B]$ consist of all lists of R, G and B

a basic primitive of some prog. languages
corresponds to strings in automata theory

$x \sqsubseteq y$ means that each symbol in x also occurs in y

e.g. $[R,R,G] \sqsubseteq [B,G,R]$!

Equivalences

If you are new to this concentrate on partial orders and read $=$ for $\equiv$

Given a preorder $\sqsubseteq$ define $\equiv$ by

$$x \equiv y \text{ iff } x \sqsubseteq y \wedge y \sqsubseteq x$$

Fact $\equiv$ is an equivalence relation

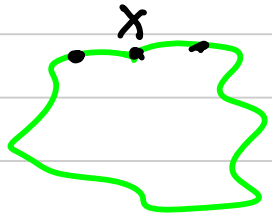
i.e. it is reflexive $(x \equiv x)$
and transitive $(x \equiv y \wedge y \equiv z \Rightarrow x \equiv z)$
and symmetric $(x \equiv y \Leftrightarrow y \equiv x)$

Fact A preorder is a partial order iff
$\equiv$ coincides with $=$

Special elements in a preordered set

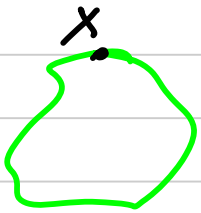
x is a maximal element iff

$$\forall y: \; y \sqsupseteq x \Rightarrow y \sqsupseteq x$$



x is a greatest element iff

$$\forall y: \; y \sqsubseteq x$$



Fact a greatest element is also maximal.

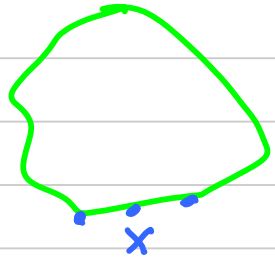
Fact in a partially ordered set the greatest element is unique if it exists

A greatest element is often written $\top$

Special elements in a preordered set

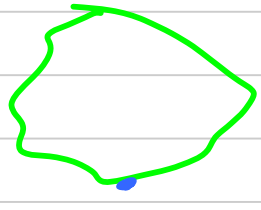
x is a minimal element iff

$$\forall y: y \sqsubseteq x \Rightarrow y \equiv x$$



x is a least element iff

$$\forall y: x \sqsubseteq y$$



Fact a least element is also minimal

Fact in a partially ordered set the least element is unique if it exists

A least element is often written $\bot$

What happened?

Essentially we turned "things on their heads"

and repeated the development.

The dual order.

Given a preorder $\sqsubseteq$ define its dual $\sqsubseteq^d$ (or $\sqsupseteq$)

by $x \sqsupseteq y$ iff $y \sqsubseteq x$.

The concepts

minimal / maximal
least / greatest
} and many more to come

are duals.

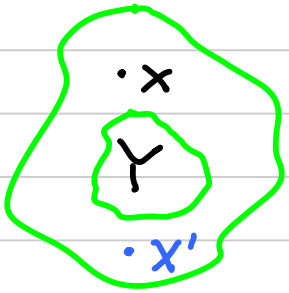
Examples

Preorder	Maximal	Greatest	Minimal	Least
$(\mathbb{Z}, \leq)$	[redacted]	[redacted]	[redacted]	[redacted]
$(BYTE, \leq)$	[redacted]	[redacted]	[redacted]	[redacted]
$\mathcal{P}\{R, G, B\}$	[redacted]	[redacted]	[redacted]	[redacted]
$LIST[R, G, B]$	[redacted]	[redacted]	[redacted]	[redacted]

Upper and lower bounds

An upper bound of a set $Y \subseteq S$ is some $x \in S$ such that $\forall y \in Y : y \sqsubseteq x$

A lower bound x' has $\forall y \in Y : x \sqsubseteq y'$



Examples

$(\mathbb{Z}, \leq)$ $\{10, 11, \ldots, 99\}$ has 99, 100, ... as upper bounds

and 10, 9, 8, ... as lower bounds

$\{3 \cdot z \mid z \in \mathbb{Z}\}$ has no upper and no lower bounds

Least upper and greatest lower bounds.

A least upper bound x of $Y \subseteq S$ is

a) an upper bound such that

b) if x' is another upper bound then $x' \sqsupseteq x$

It is written $x = \bigsqcup Y$ or $x = lub(Y)$ or $x = join(Y)$ when it exists!

A greatest lower bound x' of Y is

a) lower bound b) y' also lower bound $\Rightarrow y \sqsupseteq y'$

It is written $y = \bigsqcap Y$ or $y = glb(Y)$ or $y = meet(Y)$ when it exists!

Fact In a partially ordered set the least upper bounds and greatest lower bounds are unique if they exist.

Fact $\Updownarrow$ x is a least element of $(S, \sqsubseteq)$

x is a (greatest) lower bound of S

Fact $\Updownarrow$ x is a greatest element of $(S, \sqsubseteq)$

x is a (least) upper bound of S

Examples

$(\mathbb{Z}, \leq)$ $\bigsqcup Y = \max(Y)$ if it exists

$\bigsqcap Y = \min(Y)$ if it exists

$(BYTE, \leq)$ [redacted]

$(\mathcal{P}\{R, G, B\}, \subseteq)$ $\bigsqcup \{s_1, s_2, \ldots, s_n\} = s_1 \cup \ldots \cup s_n$ $(n > 0)$

$\bigsqcap \{s_1, s_2, \ldots, s_n\} = s_1 \cap \ldots \cap s_n$ $(n > 0)$

what if $n = 0$? [redacted]

$(\mathcal{P}\{R,G,B\}, \subseteq)$

$\bigsqcup Y = \bigcup \{y \mid y \in Y\} = y_1 \cup y_2 \cup \dots \cup y_n$ (themselves sets!)

$\bigsqcap Y = \bigcap \{y \mid y \in Y\} = y_1 \cap y_2 \cap \dots \cap y_n$

$Y \subseteq \mathcal{P}\{R,G,B\}$ if $Y = \{y_1, \dots, y_n\}$

e.g. $\bigsqcup \{\{R,G\}, \{B,R\}\} = \{R,G,B\} \in \mathcal{P}\{R,G,B\}$!

$LIST[R,G,B]$

$\bigsqcup \{L_1, \dots, L_n\} = concat(L_1, \dots, L_n)$

$\bigsqcap \{L_1, \dots, L_n\} =$ [redacted]

We dispense with the more general formula.

When certain least/greatest bounds always exist.

A partially ordered set $(S, \sqsubseteq)$ is a lattice

iff all binary subsets have least & greatest bounds.

Write $s_1 \sqcup s_2 = \bigsqcup\{s_1, s_2\}$ and $s_1 \sqcap s_2 = \bigsqcap\{s_1, s_2\}$

Some use $s_1 \oplus s_2$ Some use $s_1 \otimes s_2$

A partially ordered set $(S, \sqsubseteq)$ is a complete lattice

iff all subsets have least & greatest bounds

Is a complete lattice also a lattice? [redacted]

Examples

Partial order	Lattice	Complete Lattice
$(\mathbb{Z}, \leq)$		
$(BYTE, \leq)$		
$\mathcal{P}\{R, G, B\}$		
$LIST[R, G, B]$!		!

Properties of lattices and complete lattices.

Fact A finite & nonempty lattice is a complete lattice.

What is $\perp$? [redacted]

What is $\top$? [redacted]

Theorem For a partially ordered set the following are equivalent:

1) It is a complete lattice

2) All least upper bounds exist

3) All greatest lower bounds exist

Spelling out the properties of lub's and glb's in lattices

$$\left.\begin{array}{l} X_1 \sqsubseteq Y \\ X_2 \sqsubseteq Y \end{array}\right\} \Longleftrightarrow X_1 \sqcup X_2 \sqsubseteq Y$$

$$\left.\begin{array}{l} X \sqsubseteq Y_1 \\ X \sqsubseteq Y_2 \end{array}\right\} \Longleftrightarrow X \sqsubseteq Y_1 \sqcap Y_2$$

These are key properties in many uses of partial orders for security processes.

Exercise:

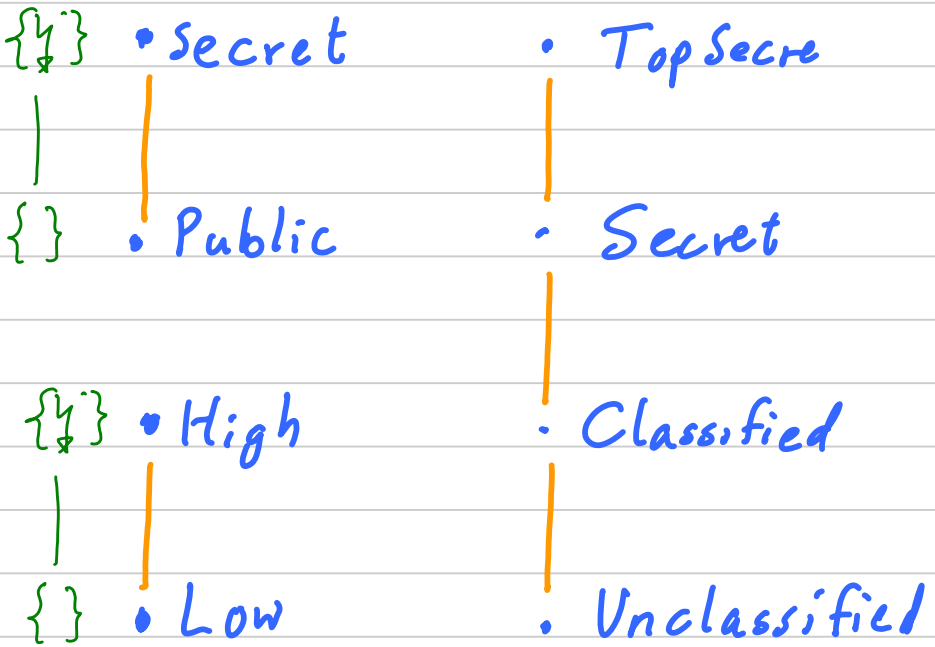
Expanding

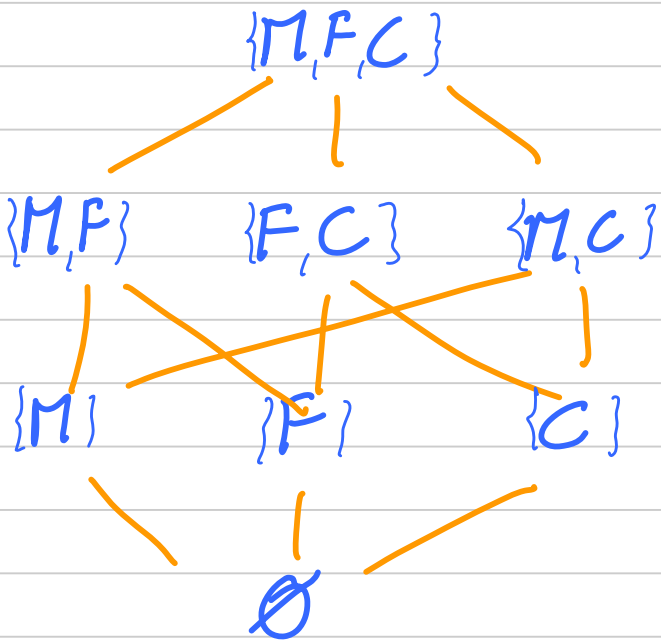
$$X_1 \sqcup X_2 \sqsubseteq Y_1 \sqcap Y_2$$

we get

[redacted]

Example Lattices for Security





$\mathcal{P}\{Med, Fin, Crim\}$

Towards an algebraic view: properties of $\sqcup$ and $\sqcap$ in lattices

Fact $x \sqsubseteq y$ iff $x \sqcup y = y$ iff $x = x \sqcap y$

Fact $\sqcup$ is idempotent $x \sqcup x \equiv x$

and commutative $x \sqcup y \equiv y \sqcup x$

and associative $(x \sqcup y) \sqcup z = x \sqcup (y \sqcup z)$

Fact Similarly for $\sqcap$

Taking the other direction

Given an operator $\varphi : S \times S \to S$

that is commutative, associative, idempotent

define $x \leq_\varphi y$ by $x \varphi y = y$

and $x \leq^\varphi y$ by $x \varphi y = x$

Fact $\leq_\varphi$ and $\leq^\varphi$ are dual partial orders

with φ as lub resp. glb

Fact $\leq_\sqcup$ is $\sqsubseteq$

Volpano's Approach to Information Flow

D. Volpano G. Smith C. Irvine JCS 96

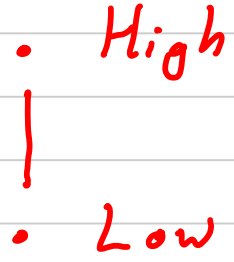
A sound type system for Secure Flow Analysis

A fairly readable account of the main ideas.
Ignore proofs (of Thm 6.8 in particular) unless you have the proper background.

Notation (us versus paper)

Security properties

$\sqsubseteq$	partial order	$\leq$
$\sqcup$	lub	$\oplus$
$\sqcap$	glb	$\otimes$

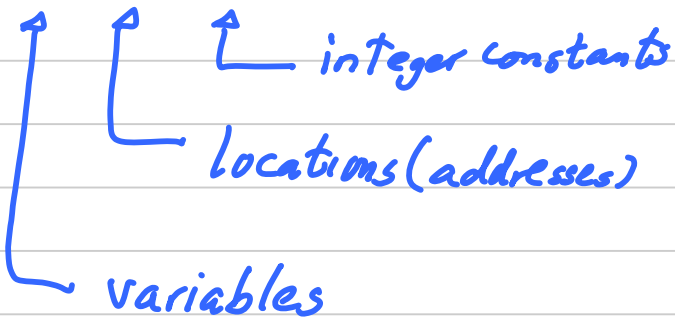


Security classification

$\underline{x}$ is the security classification of x

Programming Language

$e \in Exp \quad e ::= x \mid \ell \mid n \mid e + e' \mid e - e' \mid e = e' \mid e < e'$



$c \in Cmd \quad c ::= x := e \mid \ell := e \mid c ; c' \mid \text{if } e \text{ then } c \text{ else } c'$

$\mid \text{while } e \text{ do } c \mid \text{letvar } x := e \text{ in } c$

Types & Security Properties

$s ::= \text{High} \mid \text{Low}$

$\tau ::= s\ \text{int} \mid s\ \text{bool}$ — expressions — "reads only $\sqsubseteq s$"

$\rho ::= s\ \text{cmd}$ — commands — "assigns only $\sqsupseteq s$"

$\text{High} \cdot \quad | \quad \cdot \text{Low}$

λ is a list $[\ell_1 : \tau_1] \ldots [\ell_n : \tau_n]$

γ is a list $[x_1 : \tau_1\ \text{var}] \ldots [x_n : \tau_n\ \text{var}]$

Typing Judgements

$\lambda, \gamma \vdash e : \tau$ ensures that e reads only at levels $\sqsubseteq$ that of τ

$\lambda, \gamma \vdash c : \rho$ ensures that c assigns only at levels $\sqsupseteq$ that of ρ

note the difference

Rules for expressions

$$\lambda, \gamma \vdash x : \tau \quad \text{if } \gamma(x) = \tau \text{ var}$$

$$\lambda, \gamma \vdash l : \tau \quad \text{if } \lambda(l) = \tau$$

$$\lambda, \gamma \vdash n : s\ \text{int}$$

$$\frac{\lambda, \gamma \vdash e_1 : s\ \text{int} \qquad \lambda, \gamma \vdash e_2 : s\ \text{int}}{\lambda, \gamma \vdash e_1 < e_2 : s\ \text{bool}}$$

$$\frac{\lambda, \gamma \vdash e : \tau}{\lambda, \gamma \vdash e : \tau'} \text{ if } \tau \leq \tau'$$

$$\frac{s \leq s'}{s\ \text{int} \leq s'\ \text{int}} \qquad \frac{s \leq s'}{s\ \text{bool} \leq s'\ \text{bool}}$$

covariant subtyping

Rules for commands

τ is s int or τ is s bool
(perhaps say level(τ) = s)

$$\frac{\lambda, \gamma \vdash e : \tau}{\lambda, \gamma \vdash x := e : s\ \text{cmd}} \quad \text{if } \lambda(x) = \tau\ \text{var and } \tau \text{ is } s^{\text{int}}_{\text{bool}}$$

$$\frac{\lambda, \gamma \vdash e : \tau}{\lambda, \gamma \vdash l := e : s\ \text{cmd}} \quad \text{if } [\text{redacted}]$$

this is intuitively correct because

explicit flows [redacted]

implicit flows [redacted]

$$\frac{\lambda, \gamma \vdash c_1 : \rho \qquad \lambda, \gamma \vdash c_2 : \rho}{\lambda, \gamma \vdash c_1 ; c_2 : \rho}$$

this is intuitively correct because

explicit flows [redacted]

implicit flows [redacted]

$$\frac{\lambda, \gamma \vdash e : s\ \mathsf{bool} \qquad \lambda, \gamma \vdash c_1 : s\ \mathsf{cmd} \qquad \lambda, \gamma \vdash c_2 : s\ \mathsf{cmd}}{\lambda, \gamma \vdash \mathsf{if}\ e\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2 : s\ \mathsf{cmd}}$$

this is intuitively correct

explicit flows [redacted]

implicit flows [redacted]

recall $s\ \mathsf{int}$ means only $\exists s$ read
$s\ \mathsf{cmd}$ means only $\exists s$ written

Example

$\lambda, \gamma [x : \underline{x}\ \text{int var}][y : \underline{y}\ \text{int var}] \vdash \text{if } x = 1 \text{ then } y := 1 \text{ else } y := 2 : s\ \text{cmd}$

$\underline{x}$	$\underline{y}$	s	OK?	
High	High	High	[redacted]	1) ✓
High	High	Low	[redacted]	2)
High	Low	High	[redacted]	3)
High	Low	Low	[redacted]	4)
Low	High	High	[redacted]	5) ✓
Low	High	Low	[redacted]	6)
Low	Low	High	[redacted]	7)
Low	Low	Low	[redacted]	8) ✓

$$\frac{\lambda, \gamma \vdash c : \rho}{\lambda, \gamma \vdash c : \rho'} \text{ if } \rho \leq \rho' \qquad \frac{s \leq s'}{s' \, cmd \leq s \, cmd}$$

contravariant subtyping

this is intuitively correct because [redacted]

as a mnemonic we sometimes write

$\oplus$ int $\quad$ $\oplus$ bool $\quad$ $\ominus$ cmd

negative polarity

positive polarity

Example

$\lambda, \gamma [x : \underline{x}\ \text{int var}][y : \underline{y}\ \text{int var}] \vdash$ if $x = 1$ then $y := 1$ else $y := 2$: s cmd

$\underline{x}$	$\underline{y}$	s	OK?	
High	High	High	[redacted]	1) ✓ ⌣
High	High	Low	[redacted]	2)
High	Low	High	[redacted]	3)
High	Low	Low	[redacted]	4)
Low	High	High	[redacted]	5) ✓ ⌣
Low	High	Low	[redacted]	6) ⌣
Low	Low	High	[redacted]	7) ⌣
Low	Low	Low	[redacted]	8) ✓ ⌣

Example

$\lambda, \gamma [x : \underline{x}\ \text{int var}] \vdash$ if $x = 1$ then skip else loop : s cmd

$\underline{x}$	s	OK ?	
High	High	[redacted]	1) ✓
High	Low	[redacted]	2) ✓
Low	High	[redacted]	3) ✓
Low	Low	[redacted]	4) ✓

is this as we intend?

should we be suspicious?

$$\frac{\lambda, \gamma \vdash e : s\ \mathit{bool} \qquad \lambda, \gamma \vdash c : s\ \mathit{cmd}}{\lambda, \gamma \vdash \mathit{while}\ e\ \mathit{do}\ c : s\ \mathit{cmd}}$$

this is correct because [redacted]

should we be suspicious?

$$\frac{\lambda, \gamma \vdash e : \tau \qquad \lambda, \gamma[x : \tau\ \mathit{var}] \vdash c : s\ \mathit{cmd}}{\lambda, \gamma \vdash \mathit{letvar}\ x := e\ \mathit{in}\ c' : s\ \mathit{cmd}}$$

should we demand

$\tau = s$ $^{\mathit{int}}_{\mathit{bool}}$? [redacted]

is there a flow from e out of c'?

Some properties

If Σ is a partial order so is $\leq$ and $\sqsubseteq$.

It is possible to define an "equivalent" inference system that is syntax-directed by incorporating subtyping into the other rules; this is useful for proofs and implementations.

Exercise 1 [redacted]

Develop such "equivalent systems." (See VS1 96 Fig 5 and Thm 6.2 for ideas.)

Status

So far the development is rather comparable in its aim with respect to that of Denning.

Perhaps one issue is that it is more obvious how to implement Denning's Approach – but other papers show how to do so in general for Volpano's Approach.

The paper proceeds to develop some more convincing arguments for why the development is correct – in a sense this is the main scientific contribution of the paper: the main contribution is a non-interference result.

Background on Semantics

A program is a command that has no free variables!

A memory μ is a list $[l_1 \mapsto v_1] \ldots [l_n \mapsto v_n]$ where l_i are locations and v_i are values (integers or booleans).
It ignores variables!

A semantics clarifies

$\mu \vdash e \Rightarrow v$ expression e evaluates to v in memory μ

$\mu \vdash c \Rightarrow \mu'$ command c modifies memory μ into μ'

It can be defined formally (see VSI 96 Fig 4) or use your intuition.

First some simple correctness conditions:

Simple Security for Expressions VS196 Lem 6.3

If $\lambda, [] \vdash e : s^{int}_{bool}$ and l is in e then $\lambda(l) \leq s^{int}_{bool}$

Confinement for Commands VS196 Lem 6.4

If $\lambda, [] \vdash c : s\ cmd$ and l is assigned in c then $\lambda(l) \sqsupseteq s$

Intuitively these properties are preserved under evaluation – but this is not the direction taken in the paper.

Nonetheless, they are useful sanity checks for having got the inference rules right (which is not always easy).

Type Soundness VS196 Thm 6.8 "Non-Interference"

If $\lambda, [] \vdash c : \rho$ and $dom(\mu) = dom(\lambda) = dom(\nu)$

$\mu \vdash c \Rightarrow \mu'$

$\nu \vdash c \Rightarrow \nu'$

$\forall l : \underbrace{\lambda(l) \sqsubseteq s}_{\text{e.g. } \lambda(l) = Low} \Rightarrow \mu(l) = \nu(l)$

then $\forall l : \underbrace{\lambda(l) \sqsubseteq s}_{\text{e.g. } \lambda(l) = Low} \Rightarrow \mu'(l) = \nu'(l)$

Note: no relation between ρ and s.

This is the typical format of "no information flow" for deterministic programming languages; for non-deterministic languages a partial equivalence relation is used to show c relates to c.

Comments

Some properties of programs can be proven with respect to the semantics. Example: no buffer overflow, ...

Some properties cannot (as far as we know):

One can try to define an instrumented semantics and do the proof - this sometimes works but leaves open whether or not we got the instrumentation right. VS196 §7

Alternatively one can compare two different executions and limit the way in which information may flow. VS196 Thm 6.8
This is the preferred method for information flow as well as a number of "second-order" program analyses.

An alternative presentation of the treatment of implicit flows by Denning and Volpano and ...

```
if E1 then
    ...
    while E2 do
        ...
        if E3 then ...
        else ...
            X := E4
```

The approach of Denning, Volpano is to generate implicit flows as high in the syntax tree as possible

here we generate it as low as possible.

$\underline{E_1} \sqsubseteq \underline{X}$

$\underline{E_2} \sqsubseteq \underline{X}$

$\underline{E_3} \sqsubseteq \underline{X}$

} implicit flow

$\underline{E_4} \sqsubseteq \underline{X}$ } explicit flow