CS 591: Formal Methods in Security and Privacy

Example in Hoare Logic and Non-interference

Marco Gaboardi
gaboardi@bu.edu

Alley Stoughton
stough@bu.edu

From the previous classes

Specifications - Hoare triple

Precondition
(a logical formula)

```
Precondition
Program
Postcondition
```

$$c : P \Rightarrow Q$$

Program

Postcondition
(a logical formula)

Validity of Hoare triple

We say that the triple $c:P \Rightarrow Q$ is valid

if and only if

for every memory m such that $P(m)$

and memory m' such that $\{c\}_m = m'$

we have $Q(m')$.

Is this condition easy to check?

Semantics of Commands

This is defined on the structure of commands:

$\{\mathtt{abort}\}_m = \bot$

$\{\mathtt{skip}\}_m = m$

$\{\mathtt{x:=e}\}_m = m[x \leftarrow \{e\}_m]$

$\{\mathtt{c;c'}\}_m = \{\mathtt{c'}\}_{m'}$ If $\{\mathtt{c}\}_m = m'$

$\{\mathtt{c;c'}\}_m = \bot$ If $\{\mathtt{c}\}_m = \bot$

$\{\mathtt{if\ e\ then\ c_t\ else\ c_f}\}_m = \{\mathtt{c_t}\}_m$ If $\{e\}_m = \mathtt{true}$

$\{\mathtt{if\ e\ then\ c_t\ else\ c_f}\}_m = \{\mathtt{c_f}\}_m$ If $\{e\}_m = \mathtt{false}$

$\{\mathtt{while\ e\ do\ c}\}_m = \sup_{n \in Nat} \{\mathtt{while_n\ e\ do\ c}\}_m$

where

$\mathtt{while_n\ e\ do\ c} = \mathtt{while^n\ e\ do\ c;if\ e\ then\ abort\ else\ skip}$

and

Semantics of Commands

This is defined on the structure of commands:

$\{\texttt{abort}\}_m = \bot$

$\{\texttt{skip}\}_m = m$

$\{\texttt{x:=e}\}_m = m[x \leftarrow \{e\}_m]$

$\{\texttt{c;c'}\}_m = \{\texttt{c'}\}_{m'}$ If $\{\texttt{c}\}_m = m'$

$\{\texttt{c;c'}\}_m = \bot$ If $\{\texttt{c}\}_m = \bot$

$\{\texttt{if e then } c_t \texttt{ else } c_f\}_m = \{c_t\}_m$ If $\{e\}_m = \texttt{true}$

$\{\texttt{if e then } c_t \texttt{ else } c_f\}_m = \{c_f\}_m$ If $\{e\}_m = \texttt{false}$

$\{\texttt{while e do c}\}_m = \sup_{n \in Nat}\{\texttt{while}_n \texttt{ e do c}\}_m$

where

$\texttt{while}_n \texttt{ e do c} = \texttt{while}^n \texttt{ e do c;if e then abort else skip}$

and

$\texttt{while}^0 \texttt{ e do c} = \texttt{skip}$

$\texttt{while}^{n+1} \texttt{ e do c} = \texttt{if e then (c;while}^n \texttt{ e do c) else skip}$

Rules of Hoare Logic:

$$\frac{}{\vdash \texttt{skip}: P \Rightarrow P}$$

$$\frac{}{\vdash \texttt{x:=e} : P[e/x] \Rightarrow P}$$

$$\frac{\vdash c : P \Rightarrow R \quad \vdash c' : R \Rightarrow Q}{\vdash c;c' : P \Rightarrow Q}$$

$$\frac{P \Rightarrow S \quad \vdash c : S \Rightarrow R \quad R \Rightarrow Q}{\vdash c : P \Rightarrow Q}$$

$$\frac{}{\vdash \texttt{if } e \texttt{ then } c_1 \texttt{ else } c_2 : P \Rightarrow Q}$$

$$\frac{\vdash c : e \wedge P \Rightarrow P}{\vdash \texttt{while } e \texttt{ do } c : P \Rightarrow P \wedge \neg e}$$

Rules of Hoare Logic:

$$\frac{}{\vdash \mathtt{skip}:\ P \Rightarrow P}$$

$$\frac{}{\vdash \mathtt{x:=e}\ :\ P[e/x] \Rightarrow P}$$

$$\frac{\vdash c : P \Rightarrow R \qquad \vdash c' : R \Rightarrow Q}{\vdash c;c' :\ P \Rightarrow Q}$$

$$\frac{P \Rightarrow S \qquad \vdash c : S \Rightarrow R \qquad R \Rightarrow Q}{\vdash c :\ P \Rightarrow Q}$$

$$\frac{\vdash c_1 : e \wedge P \Rightarrow Q}{\vdash \mathtt{if}\ e\ \mathtt{then}\ c_1\ \mathtt{else}\ c_2\ :\ P \Rightarrow Q}$$

$$\frac{\vdash c\ :\ e \wedge P \Rightarrow P}{\vdash \mathtt{while}\ e\ \mathtt{do}\ c\ :\ P \Rightarrow P \wedge \neg e}$$

Rules of Hoare Logic:

$$\frac{}{\vdash \mathtt{skip}:\ P \Rightarrow P}$$

$$\frac{}{\vdash x := e\ :\ P[e/x] \Rightarrow P}$$

$$\frac{\vdash c : P \Rightarrow R \qquad \vdash c' : R \Rightarrow Q}{\vdash c;c' :\ P \Rightarrow Q}$$

$$\frac{P \Rightarrow S \qquad \vdash c : S \Rightarrow R \qquad R \Rightarrow Q}{\vdash c :\ P \Rightarrow Q}$$

$$\frac{\vdash c_1 : e \wedge P \Rightarrow Q \qquad \vdash c_2 : \neg e \wedge P \Rightarrow Q}{\vdash \mathtt{if}\ e\ \mathtt{then}\ c_1\ \mathtt{else}\ c_2\ :\ P \Rightarrow Q}$$

$$\frac{\vdash c\ :\ e \wedge P \Rightarrow P}{\vdash \mathtt{while}\ e\ \mathtt{do}\ c\ :\ P \Rightarrow P \wedge \neg e}$$

Correctness of a rule

$$\frac{\vdash c_1 : P_1 \Rightarrow Q_1 \qquad \ldots \qquad \vdash c_n : P_n \Rightarrow Q_n}{\vdash c \; : \; P \; \Rightarrow \; Q}$$

We say that a rule is correct if given valid triples as described by the assumption(s), we can prove the validity of the triple in the conclusion.

Today 1: More Hoare Logic

Another example

$\vdash$

```
x:=3;
y:=1;
while x > 1 do
  y := y+1;
  x := x-1;
```

$: \{true\} \Rightarrow \{y = 3\}$

What can be a good Invariant?

Another example

$\vdash$

```
x:=3;
y:=1;
while x > 1 do
  y := y+1;
  x := x-1;
```

$: \{true\} \Rightarrow \{y = 3\}$

What can be a good Invariant?

$\mathtt{Inv} = \{y = 4 - x \wedge x \geq 1\}$

Another example

$$\frac{}{\vdash x := 3; y := 1 : \{true\} \Rightarrow \{x = 3 \wedge 1 = 1 \wedge y = 4 - x\}}$$

Another example

$$\frac{\dfrac{\vdash x := 3 : \{true\} \Rightarrow \{x = 3\} \qquad \vdash y := 1 : \{x = 3\} \Rightarrow \{x = 3 \wedge y = 1\}}{\vdash x := 3; y := 1 : \{true\} \Rightarrow \{x = 3 \wedge y = 1\}} \qquad x = 3 \wedge y = 1 \Rightarrow x = 3 \wedge 1 = 1 \wedge y = 4 - x}{\vdash x := 3; y := 1 : \{true\} \Rightarrow \{x = 3 \wedge 1 = 1 \wedge y = 4 - x\}}$$

Another example

$$
\dfrac{
\dfrac{
\dfrac{true \Rightarrow 3 = 3 \qquad \vdash x := 3 : \{3 = 3\} \Rightarrow \{x = 3\}}{\vdash x := 3 : \{true\} \Rightarrow \{x = 3\}}
\qquad
\dfrac{x = 3 \Rightarrow x = 3 \wedge 1 = 1 \qquad \vdash y := 1 : \{x = 3 \wedge 1 = 1\} \Rightarrow \{x = 3 \wedge y = 1\}}{\vdash y := 1 : \{x = 3\} \Rightarrow \{x = 3 \wedge y = 1\}}
}{\vdash x := 3; y := 1 : \{true\} \Rightarrow \{x = 3 \wedge y = 1\}}
\qquad
x = 3 \wedge y = 1 \Rightarrow x = 3 \wedge 1 = 1 \wedge y = 4 - x
}{\vdash x := 3; y := 1 : \{true\} \Rightarrow \{x = 3 \wedge 1 = 1 \wedge y = 4 - x\}}
$$

Another example

$x = 3 \wedge y = 1 \wedge y = 4 - x \Rightarrow y = 4 - x \wedge x \geq 1$

$\vdash$
```
while x > 1 do
  y := y+1;
  x := x-1
```
$: \{y = 4 - x \wedge x \geq 1\} \Rightarrow \{y = 4 - x \wedge x = 1\}$

$y = 4 - x \wedge x = 1 \Rightarrow y = 3$

$\vdash$
```
while x > 1 do
  y := y+1;
  x := x-1
```
$: \{x = 3 \wedge y = 1 \wedge y = 4 - x\} \Rightarrow \{y = 3\}$

Another example

$\vdash$ `y := y+1; x := x-1` $: \{y = 4 - x \wedge x \geq 1 \wedge x > 1\} \Rightarrow \{y = 4 - x \wedge x \geq 1\}$

$\vdash$ `while x > 1 do: y := y+1; x := x-1` $\{y = 4 - x \wedge x \geq 1\} \Rightarrow \{y = 4 - x \wedge x \geq 1 \wedge \neg(x > 1)\}$

$\{y = 4 - x \wedge x \geq 1 \wedge \neg(x > 1)\} \Rightarrow \{y = 4 - x \wedge x = 1\}$

$x = 3 \wedge y = 1 \wedge y = 4 - x \Rightarrow y = 4 - x \wedge x \geq 1$

$\vdash$ `while x > 1 do y := y+1; x := x-1` $: \{y = 4 - x \wedge x \geq 1\} \Rightarrow \{y = 4 - x \wedge x = 1\}$

$y = 4 - x \wedge x = 1 \Rightarrow y = 3$

$\vdash$ `while x > 1 do y := y+1; x := x-1` $: \{x = 3 \wedge y = 1 \wedge y = 4 - x\} \Rightarrow \{y = 3\}$

Another example

$y = 4 - x \wedge x \geq 1 \wedge x > 1 \Rightarrow y + 1 = 4 - (x - 1) \wedge x - 1 \geq 1$

$\vdash$
```
y := y+1;
x := x-1
```
$: \{y + 1 = 4 - (x - 1) \wedge x - 1 \geq 1\} \Rightarrow \{y = 4 - x \wedge x \geq 1\}$

$\vdash$
```
y := y+1;
x := x-1
```
$: \{y = 4 - x \wedge x \geq 1 \wedge x > 1\} \Rightarrow \{y = 4 - x \wedge x \geq 1\}$

$\vdash$
```
while x > 1 do:
 y := y+1;
 x := x-1
```
$\{y = 4 - x \wedge x \geq 1\} \Rightarrow \{y = 4 - x \wedge x \geq 1 \wedge \neg(x > 1)\}$

$\{y = 4 - x \wedge x \geq 1 \wedge \neg(x > 1)\} \Rightarrow \{y = 4 - x \wedge x = 1\}$

$x = 3 \wedge y = 1 \wedge y = 4 - x \Rightarrow y = 4 - x \wedge x \geq 1$

$\vdash$
```
while x > 1 do
 y := y+1;
 x := x-1
```
$: \{y = 4 - x \wedge x \geq 1\} \Rightarrow \{y = 4 - x \wedge x = 1\}$

$y = 4 - x \wedge x = 1 \Rightarrow y = 3$

$\vdash$
```
while x > 1 do
 y := y+1;
 x := x-1
```
$: \{x = 3 \wedge y = 1 \wedge y = 4 - x\} \Rightarrow \{y = 3\}$

Another example

$\vdash$ `y := y+1` $: \{y+1 = 4-(x-1) \wedge x-1 \geq 1\} \Rightarrow \{y = 4-(x-1) \wedge x-1 \geq 1\}$

$\vdash$ `x := x-1` $: \{y = 4-(x-1) \wedge x-1 \geq 1\} \Rightarrow \{y = 4-x \wedge x \geq 1\}$

$y = 4-x \wedge x \geq 1 \wedge x > 1 \Rightarrow y+1 = 4-(x-1) \wedge x-1 \geq 1$

$\vdash$
```
y := y+1;
x := x-1
```
$: \{y+1 = 4-(x-1) \wedge x-1 \geq 1\} \Rightarrow \{y = 4-x \wedge x \geq 1\}$

$\vdash$
```
y := y+1;
x := x-1
```
$: \{y = 4-x \wedge x \geq 1 \wedge x > 1\} \Rightarrow \{y = 4-x \wedge x \geq 1\}$

$\vdash$
```
while x > 1 do:
  y := y+1;
  x := x-1
```
$\{y = 4-x \wedge x \geq 1\} \Rightarrow \{y = 4-x \wedge x \geq 1 \wedge \neg(x > 1)\}$

$\{y = 4-x \wedge x \geq 1 \wedge \neg(x > 1)\} \Rightarrow \{y = 4-x \wedge x = 1\}$

$x = 3 \wedge y = 1 \wedge y = 4-x \Rightarrow y = 4-x \wedge x \geq 1$

$\vdash$
```
while x > 1 do
  y := y+1;
  x := x-1
```
$: \{y = 4-x \wedge x \geq 1\} \Rightarrow \{y = 4-x \wedge x = 1\}$

$y = 4-x \wedge x = 1 \Rightarrow y = 3$

$\vdash$
```
while x > 1 do
  y := y+1;
  x := x-1
```
$: \{x = 3 \wedge y = 1 \wedge y = 4-x\} \Rightarrow \{y = 3\}$

Another example

$$\dfrac{\vdash \begin{array}{l}\texttt{x :=3;}\\ \texttt{y :=1;}\end{array} : \{true\} \Rightarrow \{x = 3 \land 1 = 1 \land y = 4 - x\} \qquad \vdash \begin{array}{l}\texttt{while x > 1 do}\\ \quad\texttt{y := y+1;}\\ \quad\texttt{x := x-1;}\end{array} : \{x = 3 \land y = 1 \land y = 4 - x\} \Rightarrow \{y = 3\}}{\vdash \begin{array}{l}\texttt{x:=3;}\\ \texttt{y:=1;}\\ \texttt{while x > 1 do}\\ \quad\texttt{y := y+1;}\\ \quad\texttt{x := x-1;}\end{array} : \{true\} \Rightarrow \{y = 3\}}$$

What happens if the loop does not end?

Another example

$$\vdash \boxed{\texttt{while true do skip}} : \{true\} \Rightarrow \{false\}$$

Can we prove it?

Another example

$$\vdash \boxed{\texttt{while true do skip}} : \{true\} \Rightarrow \{false\}$$

Can we prove it?

What can be a good Invariant?

Another example

$$\vdash \boxed{\texttt{while true do skip}} : \{true\} \Rightarrow \{false\}$$

Can we prove it?

What can be a good Invariant?

$$\texttt{Inv} = \{true\}$$

Another example

$$\vdash \boxed{\texttt{while true do skip}} : \{true\} \Rightarrow \{false\}$$

$$\frac{\vdash \texttt{c} : \texttt{true} \wedge \texttt{true} \Rightarrow \texttt{true}}{\vdash \texttt{while true do skip} : \texttt{true} \Rightarrow \texttt{true} \wedge \neg\texttt{true}}$$

Partial vs Total correctness

Partial correctness: the definition of validity requires the postcondition to hold only if the program terminates.

Total correctness: the definition of validity requires the program to terminate and the postcondition to hold.

Total Correctness
Validity of Hoare triple

We say that the triple `c:P⇒Q` is valid

if and only if

for every memory `m` such that `P(m)`

there exists a memory m' such that

$\{c\}_m = m'$ and `Q(m')`.

Total Correctness Hoare Logic

- All the rules except the ones for abort and while support total correctness.
- We could give a total correctness rule for while.

How do we know that these are the right rules?

Soundness

If we can derive $\vdash \texttt{c : P} \Rightarrow \texttt{Q}$ through the rules of the logic, then the triple $\texttt{c : P} \Rightarrow \texttt{Q}$ is valid.

Are the rules we presented sound?

Completeness

If a triple $\texttt{c} : \texttt{P} \Rightarrow \texttt{Q}$ is valid, then we can derive $\vdash \texttt{c} : \texttt{P} \Rightarrow \texttt{Q}$ through the rules of the logic.

Are the rules we presented complete?

Relative Completeness

$$\frac{P \Rightarrow S \qquad \vdash \texttt{c} : S \Rightarrow R \qquad R \Rightarrow Q}{\vdash \texttt{c} : P \Rightarrow Q}$$

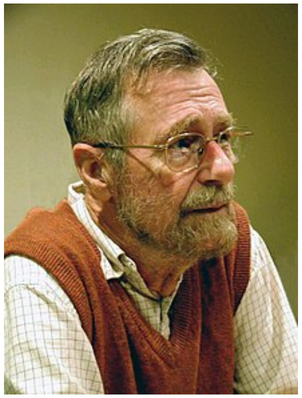
Relative Completeness

$$\frac{P \Rightarrow S \qquad \vdash c : S \Rightarrow R \qquad R \Rightarrow Q}{\vdash c : \ P \Rightarrow Q}$$

If a triple $c : Pre \Rightarrow Post$ is valid, and we have an oracle to derive all the true statements of the form $P \Rightarrow S$ and of the form $R \Rightarrow Q$,which we can use in applications of the conseq rule, then we can derive $\vdash c : Pre \Rightarrow Post$ through the rules of the logic.

Today 2: weakest precondition calculus

Predicate Transformer Semantics



Given a program `c` and an assertion `P` we can define an assertion `wp(c,P)` which is the weakest precondition of `c` and `P`, i.e. `c: wp(c,P)⇒P` is a valid triple, and for every triple `c:Q⇒P` we have `Q⇒wp(c,P)`

Weakest precondition

This is defined on the structure of commands:

$$\mathtt{wp(abort,P)} = \mathtt{false}$$

$$\mathtt{wp(skip,P)} = \mathtt{P}$$

$$\mathtt{wp(x:=e,P)} = \mathtt{P[x \leftarrow \{e\}_m]}$$

$$\mathtt{wp(c;c',P)} = \mathtt{wp(c,wp(c',P))}$$

$$\mathtt{wp(if\ e\ then\ c_t\ else\ c_f,P)} = (\mathtt{e} \Rightarrow \mathtt{wp(c_t,P)}) \land (\neg\mathtt{e} \Rightarrow \mathtt{wp(c_t,P)})$$

$$\mathtt{wp(while\ e\ do\ c,P)} = \exists_{n \in \mathtt{Nat}}\ \mathtt{P}_n \text{ where}$$

Weakest precondition

This is defined on the structure of commands:

$$\texttt{wp(abort,P)} = \texttt{false}$$

$$\texttt{wp(skip,P)} = \texttt{P}$$

$$\texttt{wp(x:=e,P)} = \texttt{P[x}\leftarrow\texttt{\{e\}}_{\texttt{m}}\texttt{]}$$

$$\texttt{wp(c;c',P)} = \texttt{wp(c,wp(c',P))}$$

$$\texttt{wp(if e then c}_{\texttt{t}}\texttt{ else c}_{\texttt{f}}\texttt{,P)} = \texttt{(e} \Rightarrow \texttt{wp(c}_{\texttt{t}}\texttt{,P))} \wedge \texttt{(}\neg\texttt{e} \Rightarrow \texttt{wp(c}_{\texttt{t}}\texttt{,P))}$$

$$\texttt{wp(while e do c,P)} = \exists_{\texttt{n}\in\texttt{Nat}}\, \texttt{P}_{\texttt{n}} \text{ where}$$

$$\texttt{P}_{\texttt{0}} = \neg\texttt{e} \wedge \texttt{P}$$

$$\texttt{P}_{\texttt{n+1}} = \texttt{e} \wedge \texttt{wp(c,P}_{\texttt{n}}\texttt{)}$$

Today 3: security as information flow control

Some Examples of Security Properties

- Access Control
- Encryption
- Malicious Behavior Detection
- Information Filtering
- Information Flow Control

Some Examples of Security Properties

- Access Control
- Encryption
- Malicious Behavior Detection
- Information Filtering
- Information Flow Control

Private vs Public

We want to distinguish confidential information that need to be kept secret from nonconfidential information that can be accessed by everyone.

We assume that every variable is tagged with one either public or private.

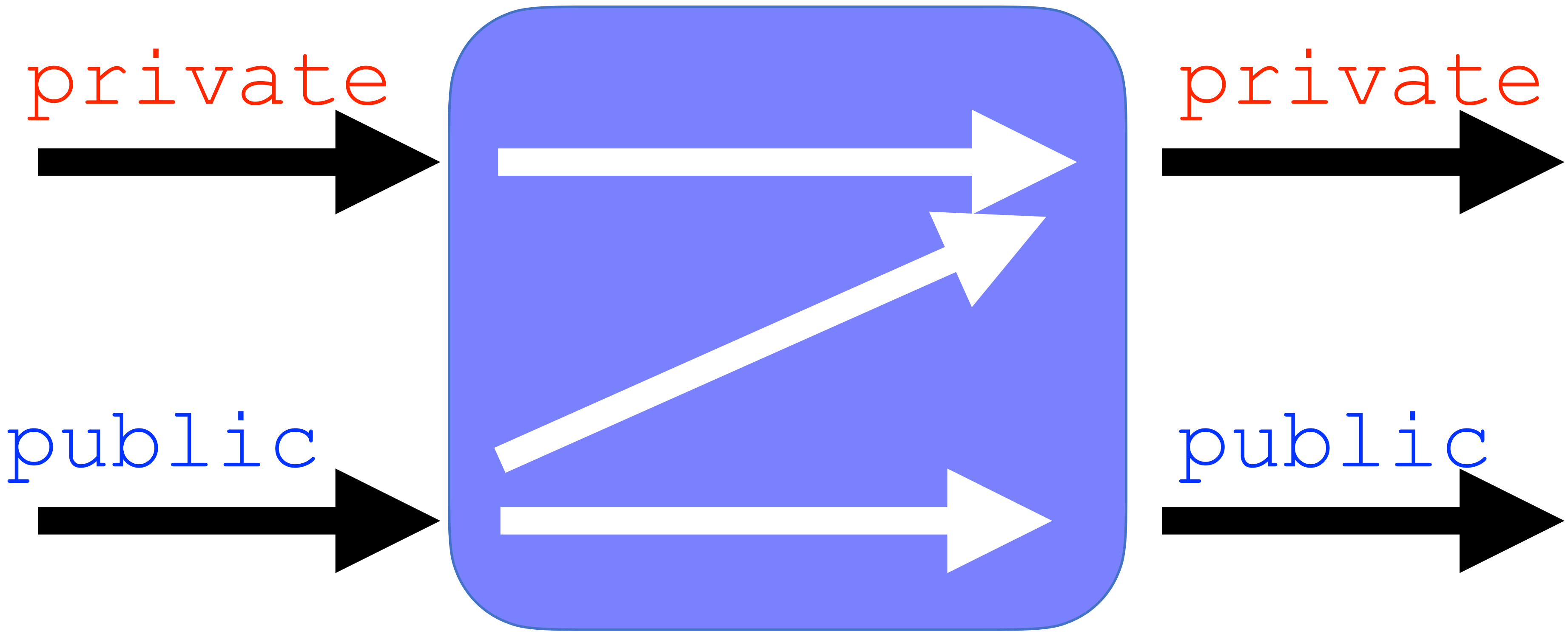
`x:public` `x:private`

Information Flow Control

We want to guarantee that confidential information do not flow in what is considered nonconfidential.

Information Flow Control

We want to guarantee that confidential information do not flow in what is considered nonconfidential.



Is this program secure?

```
x:private
y:public

x:=y
```

Is this program secure?

```
x:private
y:public

x:=y
```

Secure

Is this program secure?

```
x:private
y:public

y:=x
```

Is this program secure?

```
x:private
y:public

y:=x
```

Insecure

Is this program secure?

```
x:private
y:public

y:=x;
y:=5
```

Is this program secure?

```
x:private
y:public

y:=x;
y:=5
```

Secure

Is this program secure?

```
x:private
y:public

if y mod 3 = 0 then
  x:=1
else
  x:=0
```

Is this program secure?

```
x:private
y:public

if y mod 3 = 0 then
  x:=1
else
  x:=0
```

Secure

Is this program secure?

```
x:private
y:public

if x mod 3 = 0 then
  y:=1
else
  y:=0
```

Is this program secure?

```
x:private
y:public

if x mod 3 = 0 then
  y:=1
else
  y:=0
```

Insecure

How can we formulate a policy that forbids flows from private to public?

Low equivalence

Two memories m_1 and m_2 are low equivalent if and only if they coincide in the value that they assign to public variables.

In symbols: $m_1 \sim_{low} m_2$

Noninterference

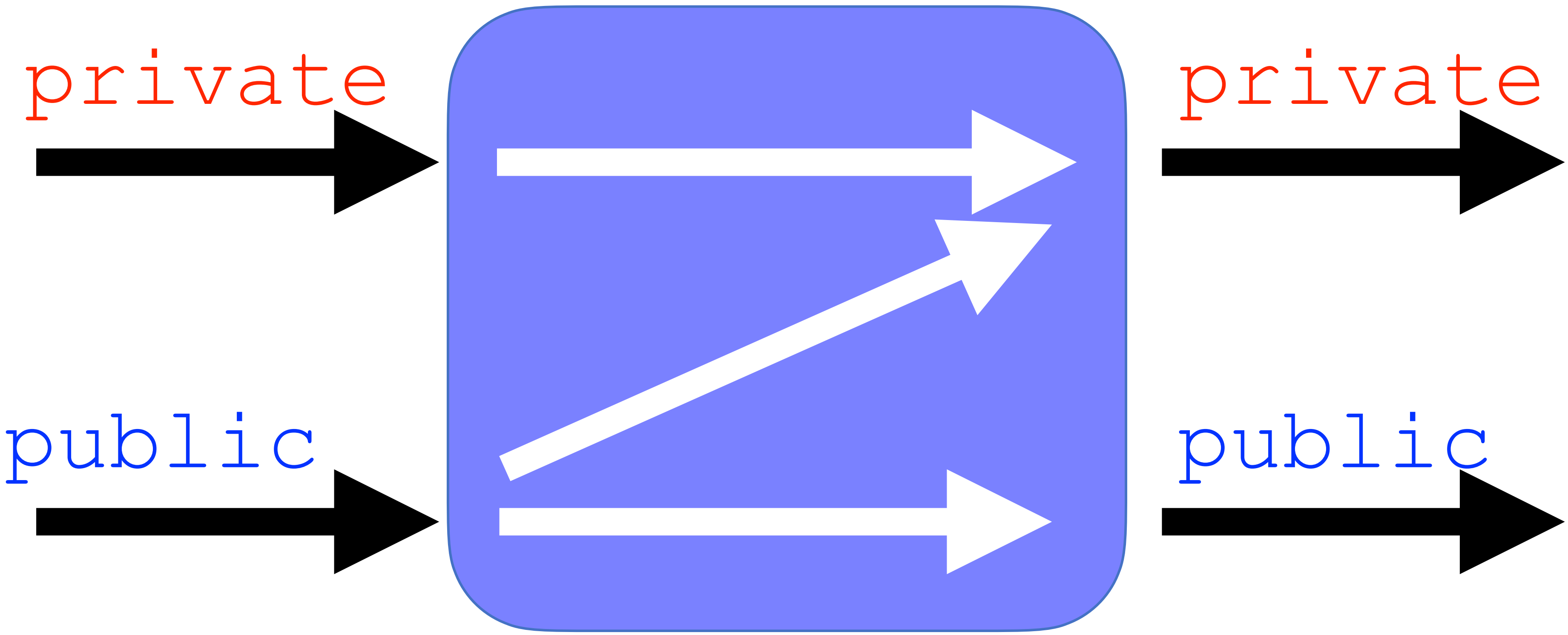
A program `prog` is noninterferent if and only if, whenever we run it on two memories m_1 and m_2 that are low equivalent, we obtain two memories m_1' and m_2' which are also low equivalent.

Noninterference

In symbols

$m_1 \sim_{low} m_2$ and $\{c\}_{m1} = m_1'$ and $m_2' \{c\}_{m2} = m_2'$

implies $m_1' \sim_{low} m_2'$



Does this program satisfy noninterference?

```
x:private
y:public


x:=y
```

Does this program satisfy noninterference?

```
x:private
y:public

x:=y
```

Yes

Does this program satisfy noninterference?

```
x:private
y:public

x:=y
```

Yes

$m^{in}{}_1=[x=n_1,y=k]$

Does this program satisfy noninterference?

```
x:private
y:public

x:=y
```

Yes

$m^{in}{}_1=[x=n_1,y=k]$

$m^{in}{}_2=[x=n_2,y=k]$

Does this program satisfy noninterference?

```
x:private
y:public

x:=y
```

Yes

m^{in}_1=[x=n_1,y=k]

m^{out}_1=[x=k,y=k]

m^{in}_2=[x=n_2,y=k]

m^{out}_2=[x=k,y=k]

Does this program satisfy noninterference?

```
x:private
y:public

y:=x
```

Does this program satisfy noninterference?

```
x:private
y:public

y:=x
```

$\boxed{\text{No}}$

Does this program satisfy noninterference?

```
x:private
y:public

y:=x
```

$\boxed{\text{No}}$

$m^{in}_1=[x=n_1,y=k]$

Does this program satisfy noninterference?

```
x:private
y:public

y:=x
```

$\boxed{\text{No}}$

$m^{in}_1=[x=n_1,y=k]$

$m^{in}_2=[x=n_2,y=k]$

Does this program satisfy noninterference?

```
x:private
y:public

y:=x
```

No

$m^{in}_1=[x=n_1,y=k]$

$m^{out}_1=[x=n_1,y=n_1]$

$m^{in}_2=[x=n_2,y=k]$

$m^{out}_2=[x=n_2,y=n_2]$

Does this program satisfy noninterference?

```
x:private
y:public

y:=x
y:=5
```

Does this program satisfy noninterference?

```
x:private
y:public

y:=x
y:=5
```

Yes

Does this program satisfy noninterference?

```
x:private
y:public

y:=x
y:=5
```

Yes

m^{in}_1=[x=n_1,y=k]

Does this program satisfy noninterference?

```
x:private
y:public

y:=x
y:=5
```

Yes

m^{in}_1=[x=n_1,y=k]

m^{in}_2=[x=n_2,y=k]

Does this program satisfy noninterference?

```
x:private
y:public

y:=x
y:=5
```

Yes

$m^{in}_1=[x=n_1,y=k]$

$m^{out}_1=[x=n_1,y=5]$

$m^{in}_2=[x=n_2,y=k]$

$m^{out}_2=[x=n_2,y=5]$

Does this program satisfy noninterference?

```
x:private
y:public
if y mod 3 = 0 then
  x:=1
else
  x:=0
```

Does this program satisfy noninterference?

```
x:private
y:public
if y mod 3 = 0 then
  x:=1
else
  x:=0
```

Yes

Does this program satisfy noninterference?

```
x:private
y:public
if y mod 3 = 0 then
  x:=1
else
  x:=0
```

Yes

$m^{in}_1=[x=n_1,y=6]$

Does this program satisfy noninterference?

```
x:private
y:public
if y mod 3 = 0 then
  x:=1
else
  x:=0
```

Yes

m^{in}_1=[x=n_1,y=6] m^{in}_2=[x=n_2,y=6]

Does this program satisfy noninterference?

```
x:private
y:public
if y mod 3 = 0 then
  x:=1
else
  x:=0
```

Yes

$m^{in}_1=[x=n_1,y=6]$ $m^{in}_2=[x=n_2,y=6]$

$m^{out}_1=[x=1,y=6]$ $m^{out}_2=[x=1,y=6]$

Does this program satisfy noninterference?

```
x:private
y:public
if x mod 3 = 0 then
  y:=1
else
  y:=0
```

Does this program satisfy noninterference?

```
x:private
y:public
if x mod 3 = 0 then
  y:=1
else
  y:=0
```

$\boxed{\text{No}}$

Does this program satisfy noninterference?

```
x:private
y:public
if x mod 3 = 0 then
  y:=1
else
  y:=0
```

No

m^{in}_1=[x=6,y=k]

Does this program satisfy noninterference?

```
x:private
y:public
if x mod 3 = 0 then
  y:=1
else
  y:=0
```

No

m^{in}_1=[x=6,y=k]

m^{in}_2=[x=5,y=k]

Does this program satisfy noninterference?

```
x:private
y:public
if x mod 3 = 0 then
  y:=1
else
  y:=0
```

No

m^{in}_1=[x=6,y=k]

m^{in}_2=[x=5,y=k]

m^{out}_1=[x=6,y=1]

m^{out}_2=[x=5,y=0]

Does this program satisfy noninterference?

```
s1:public
s2:private
r:private
i:public

proc Compare (s1:list[n] bool,s2:list[n] bool)
i:=0;
r:=0;
while i<n /\ r=0 do
 if not(s1[i]=s2[i]) then
    r:=1
 i:=i+1
```

Does this program satisfy noninterference?

```
s1:public
s2:private
r:private
i:public

proc Compare (s1:list[n] bool,s2:list[n] bool)
i:=0;
r:=0;
while i<n /\ r=0 do
 if not(s1[i]=s2[i]) then
     r:=1
 i:=i+1
```

No

How can we prove our programs noninterferent?