Formal Reasoning for Security and Privacy
Lecture 2

Marco Gaboardi
Boston University
gaboardi@bu.edu

Formal Semantics

We need to assign a formal meaning to the different components:

formal semantics of specification conditions

```
Precondition
Program
Postcondition
```

formal semantics of programs

formal semantics of specification conditions

We also need to describe the rules which combine program and specifications.

Programming Language

```
c::= abort
   | skip
   | x:=e
   | c;c
   | if e then c else c
   | while e do c
```

x, y, z, ...	program variables
$e_1, e_2, \ldots$	expressions
$c_1, c_2, \ldots$	commands

Semantics of Commands

This is defined on the structure of commands:

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

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

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

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

$$\{c;c'\}_m = \bot \qquad \text{If} \qquad \{c\}_m = \bot$$

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

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

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

where

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

and

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

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

Hoare triple

```
Precondition
Program
Postcondition
```

Precondition
(a logical formula)

$$c : P \Rightarrow Q$$

Program

Postcondition
(a logical formula)

Rules of Hoare Logic:

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

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

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

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

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

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

Information Flow Control
And
Relational Hoare Logic
(RHL)

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.

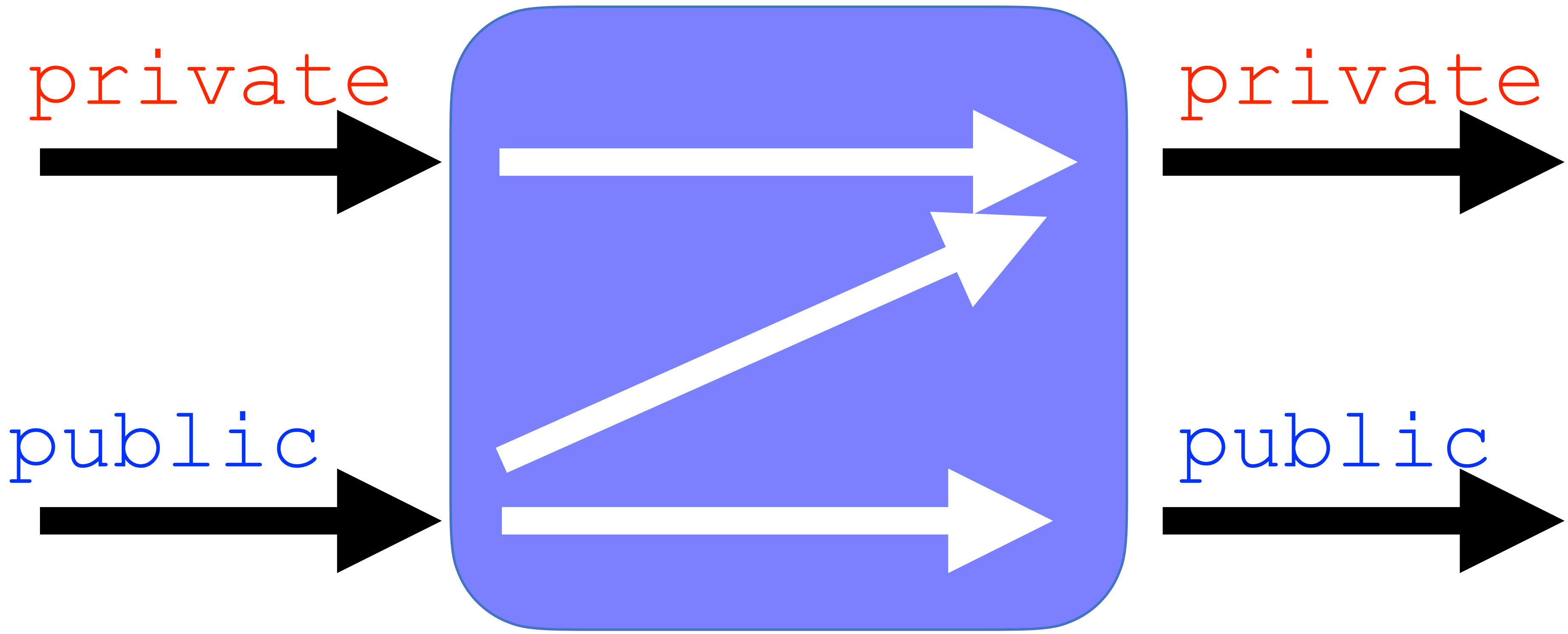
`x:public` `x:private`

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

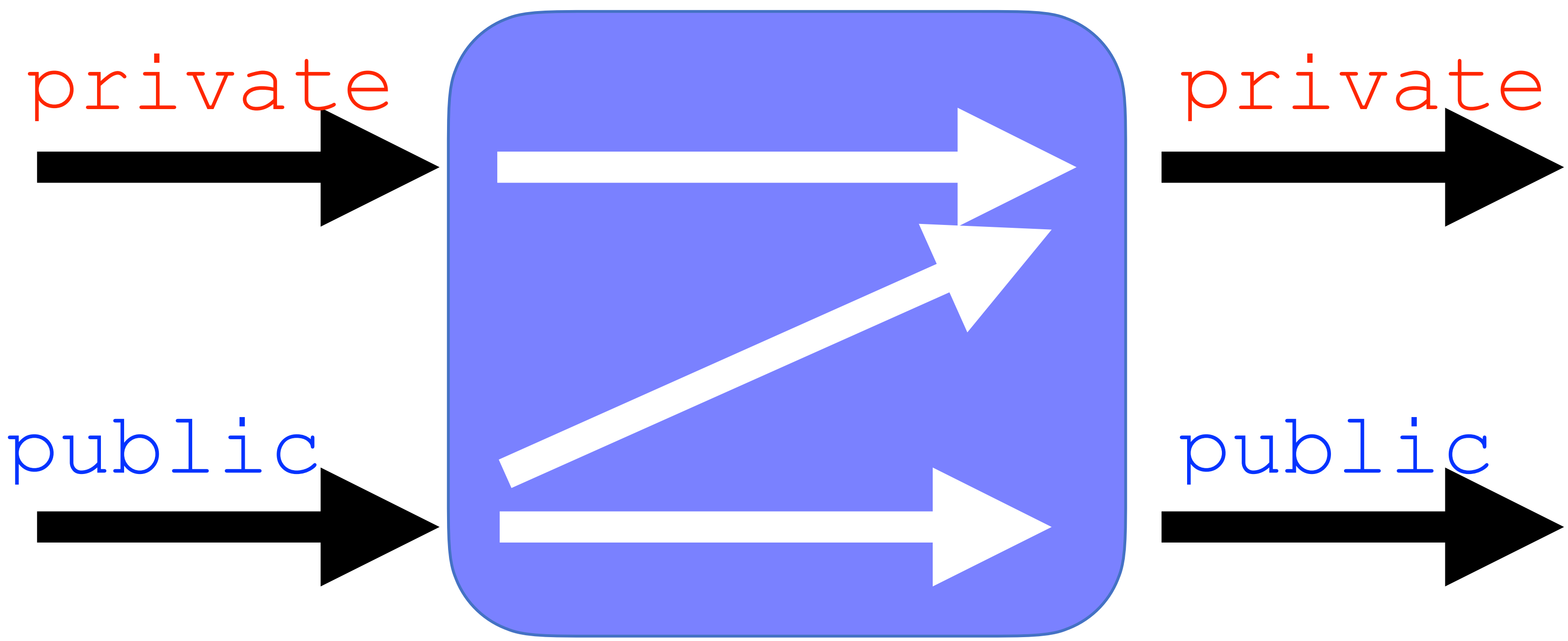
A program `prog` is noninterferent if and only if, whenever we run it on two low equivalent memories m_1 and m_2 we have that:

1) Either both terminate or both non-terminate
2) If they both terminate we obtain two low equivalent memories m_1' and m_2'.

Noninterference

In symbols, c is noninterferent if and only if for every $m_1 \sim_{low} m_2$:

1) $\{c\}_{m1} = \perp$ iff $\{c\}_{m2} = \perp$
2) $\{c\}_{m1} = m_1'$ and $\{c\}_{m2} = m_2'$ 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
```

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
```

No

Does this program satisfy noninterference?

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

$\boxed{\text{No}}$

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

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

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

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?

Noninterference

In symbols, c is noninterferent if and only if for every $m_1 \sim_{low} m_2$:

1) $\{c\}_{m1}=\perp$ iff $\{c\}_{m2}=\perp$
2) $\{c\}_{m1}=m_1'$ and $\{c\}_{m2}=m_2'$ implies $m_1' \sim_{low} m_2'$

Is this condition easy to check?

Can we use the tool we studied so far?

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 $\texttt{c:P}\Rightarrow\texttt{Q}$ is valid

if and only if

for every memory $\texttt{m}$ such that $\texttt{P(m)}$

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

we have $\texttt{Q(m')}$.

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)`

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

we have `Q(m')`.

Validity talks only about one memory. How can we manage two memories?

Relational Property

In symbols, c is noninterferent if and only if for every $m_1 \sim_{low} m_2$:

1) $\{c\}_{m1} = \perp$ iff $\{c\}_{m2} = \perp$

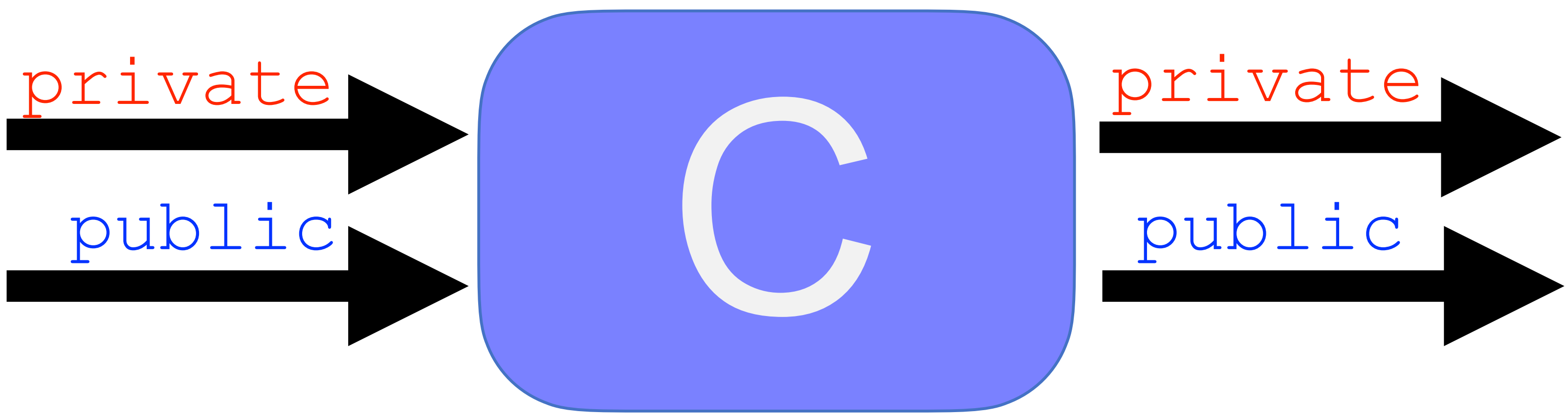
2) $\{c\}_{m1} = m_1'$ and $\{c\}_{m2} = m_2'$ implies $m_1' \sim_{low} m_2'$

Relational Property

In symbols, c is noninterferent if and only if for every $m_1 \sim_{low} m_2$:

1) $\{c\}_{m1}=\perp$ iff $\{c\}_{m2}=\perp$

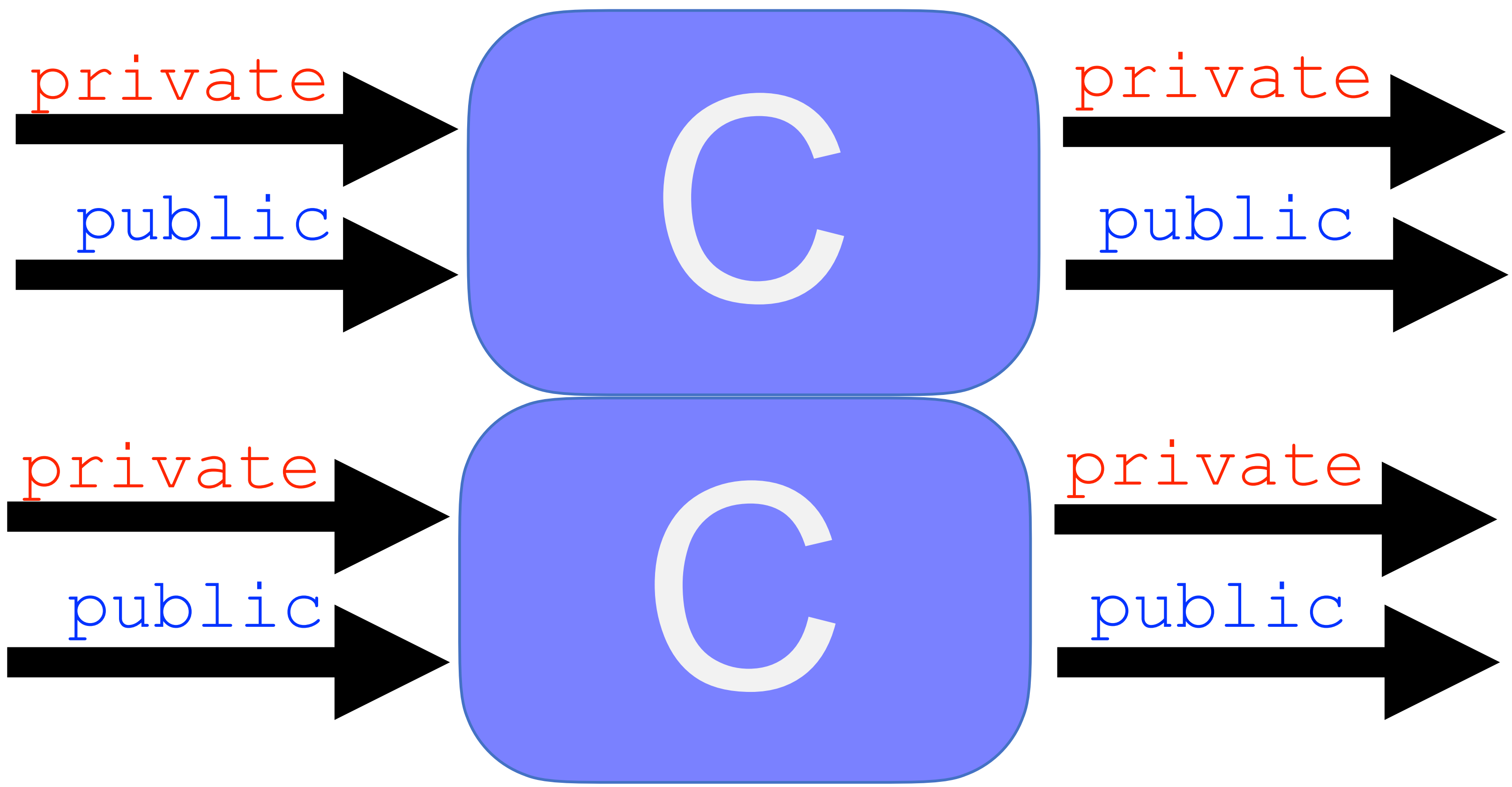
2) $\{c\}_{m1}=m_1'$ and $\{c\}_{m2}=m_2'$ implies $m_1' \sim_{low} m_2'$



Relational Property

In symbols, c is noninterferent if and only if for every $m_1 \sim_{low} m_2$:

1) $\{c\}_{m1} = \perp$ iff $\{c\}_{m2} = \perp$
2) $\{c\}_{m1} = m_1'$ and $\{c\}_{m2} = m_2'$ implies $m_1' \sim_{low} m_2'$

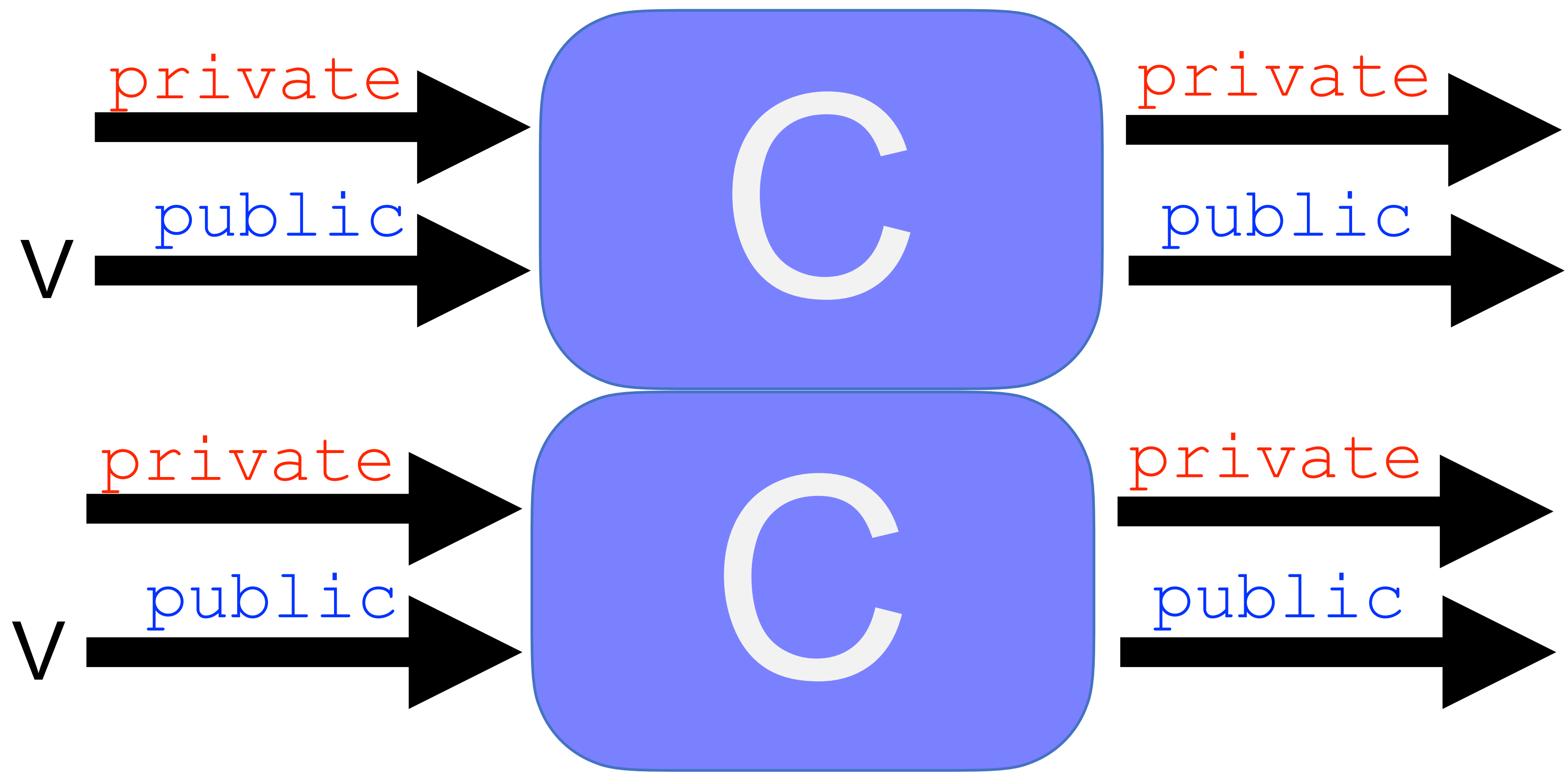


Relational Property

In symbols, c is noninterferent if and only if for every $m_1 \sim_{low} m_2$:

1) $\{c\}_{m1} = \perp$ iff $\{c\}_{m2} = \perp$

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

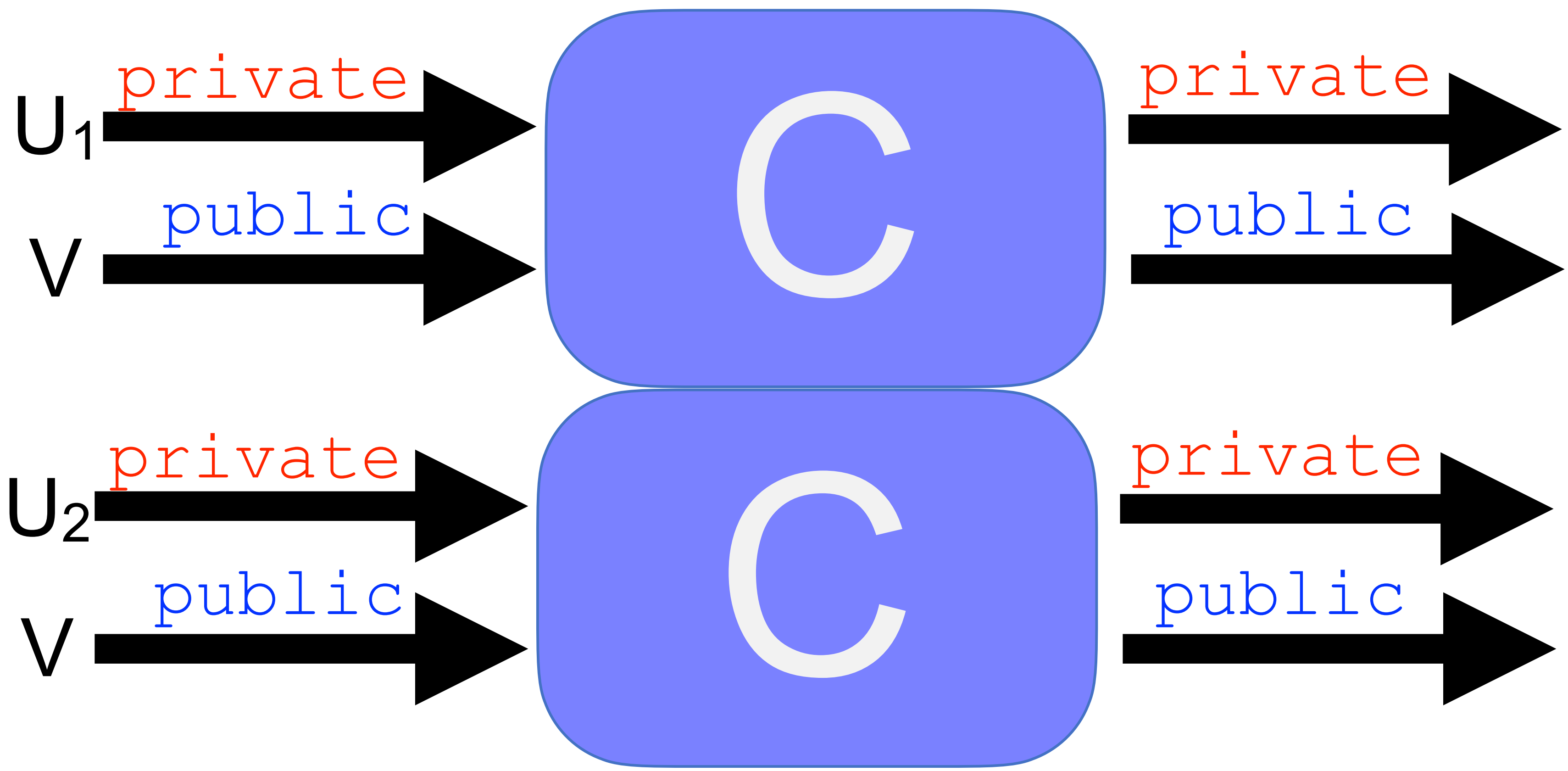


Relational Property

In symbols, c is noninterferent if and only if for every $m_1 \sim_{low} m_2$:

1) $\{c\}_{m1}=\perp$ iff $\{c\}_{m2}=\perp$

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

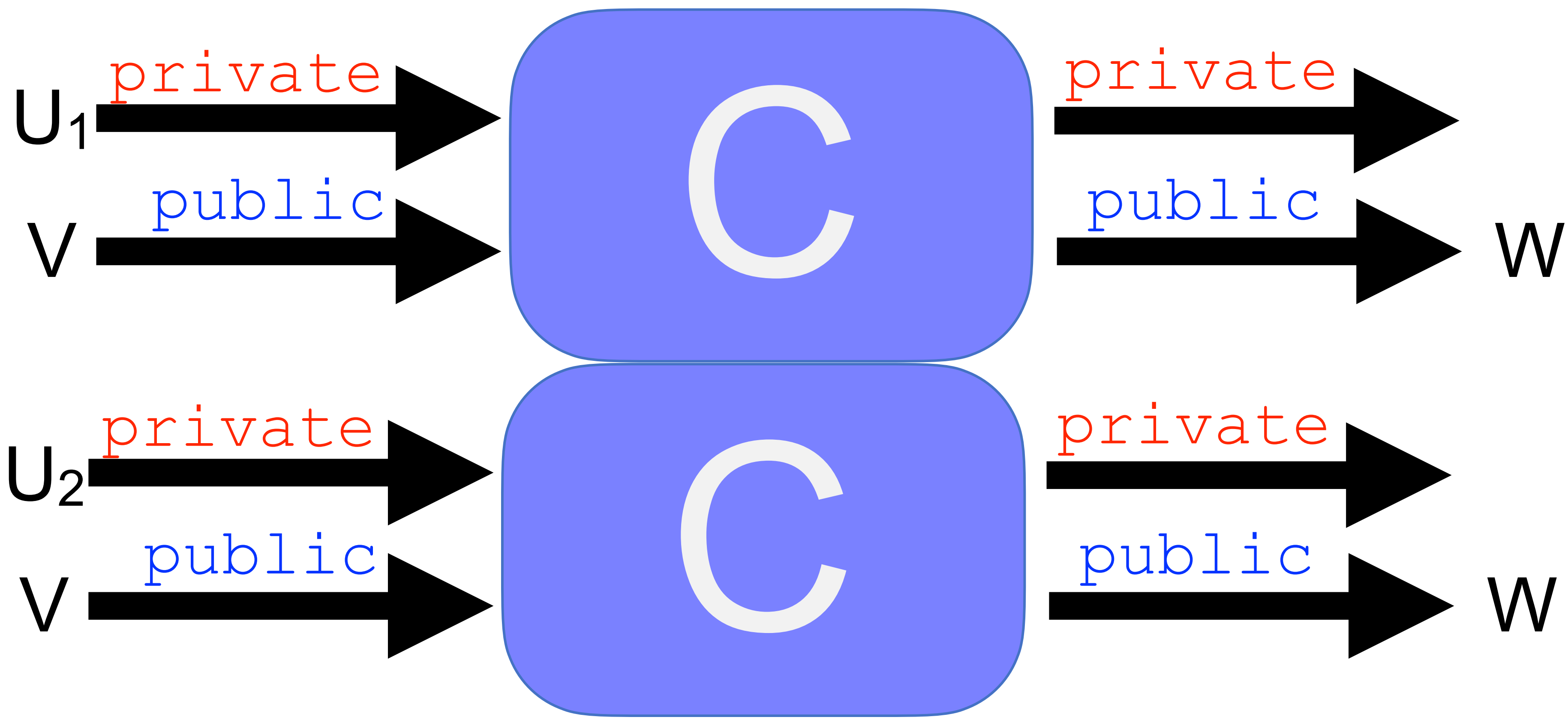


Relational Property

In symbols, c is noninterferent if and only if for every $m_1 \sim_{low} m_2$:

1) $\{c\}_{m1}=\perp$ iff $\{c\}_{m2}=\perp$

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

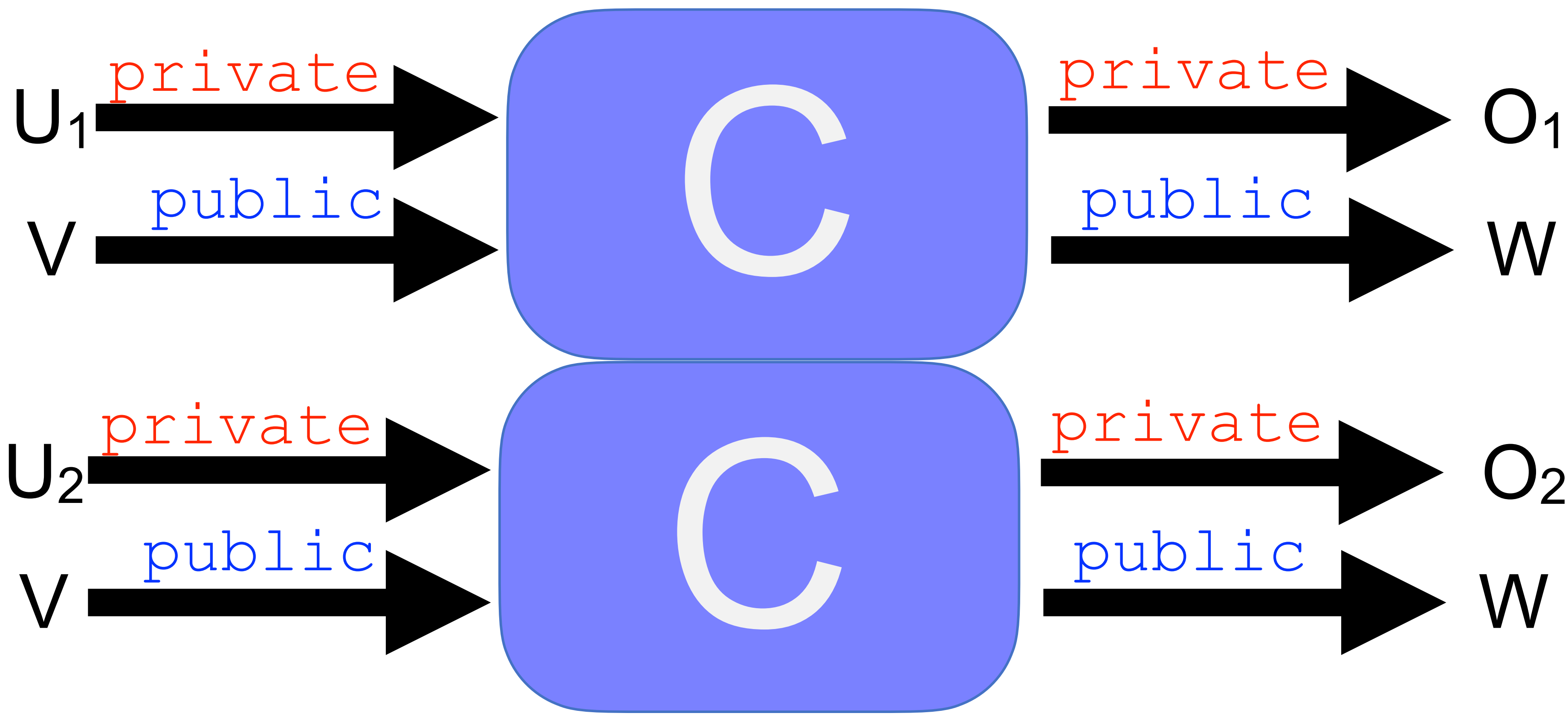


Relational Property

In symbols, c is noninterferent if and only if for every $m_1 \sim_{low} m_2$:

1) $\{c\}_{m1} = \bot$ iff $\{c\}_{m2} = \bot$

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



Relational Hoare Logic - RHL

Precondition
(a logical formula)

Precondition
$\texttt{Program}_1 \sim \texttt{Program}_2$
Postcondition

$$c_1 \sim c_2 : P \Rightarrow Q$$

Program

Program

Postcondition
(a logical formula)

Relational Assertions

$$c_1 \sim c_2 : P \Rightarrow Q$$

Need to talk about variables
of the two memories

Relational Assertions

$$c_1 \sim c_2 : P \Rightarrow Q$$

Need to talk about variables
of the two memories

$$c_1 \sim c_2 : x\langle 1\rangle \leq x\langle 2\rangle \Rightarrow x\langle 1\rangle \geq x\langle 2\rangle$$

Relational Assertions

$$c_1 \sim c_2 : P \Rightarrow Q$$

Need to talk about variables
of the two memories

$$c_1 \sim c_2 : x\langle 1 \rangle \leq x\langle 2 \rangle \Rightarrow x\langle 1 \rangle \geq x\langle 2 \rangle$$

Tags describing which
memory we are referring to.

Validity of Hoare quadruple

We say that the quadruple $c_1 \sim c_2 : P \Rightarrow Q$ is valid if and only if for every pair of memories m_1, m_2 such that $P(m_1, m_2)$ we have:

1) $\{c_1\}_{m1} = \perp$ iff $\{c_2\}_{m2} = \perp$
2) $\{c_1\}_{m1} = m_1'$ and $\{c_2\}_{m2} = m_2'$ implies $Q(m_1', m_2')$.

Validity of Hoare quadruple

We say that the quadruple $c_1 \sim c_2 : P \Rightarrow Q$ is valid if and only if for every pair of memories m_1, m_2 such that $P(m_1, m_2)$ we have:

1) $\{c_1\}_{m1} = \perp$ iff $\{c_2\}_{m2} = \perp$
2) $\{c_1\}_{m1} = m_1'$ and $\{c_2\}_{m2} = m_2'$ implies $Q(m_1', m_2')$.

Is this easy to check?

Rules of Relational Hoare Logic

Skip

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

Correctness of an axiom

We say that an axiom is correct if we can prove the validity of each instance of the conclusion.

Correctness of an axiom

We say that an axiom is correct if we can prove the validity of each instance of the conclusion.

Is this still good for RHL?

Correctness of Skip Rule

$$\dfrac{}{\vdash \texttt{skip} \sim \texttt{skip} : P \Rightarrow P}$$

To show this rule correct we need to show the validity of the quadruple `skip~skip: P⇒P`.

Correctness of Skip Rule

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

To show this rule correct we need to show the validity of the quadruple $\texttt{skip} \sim \texttt{skip}: P \Rightarrow P$.

For every m_1, m_2 such that $P(m_1, m_2)$ and m_1', m_2' such that $\{\texttt{skip}\}_{m1} = m_1'$ and $\{\texttt{skip}\}_{m2} = m_2'$ we need $P(m_1', m_2')$.

Correctness of Skip Rule

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

To show this rule correct we need to show the validity of the quadruple $\texttt{skip} \sim \texttt{skip}: P \Rightarrow P$.

For every m_1, m_2 such that $P(m_1, m_2)$ and m_1', m_2' such that $\{\texttt{skip}\}_{m1} = m_1'$ and $\{\texttt{skip}\}_{m2} = m_2'$ we need $P(m_1', m_2')$.

Follow easily by our semantics:
$\{\texttt{skip}\}_m = m$

Rules of Relational Hoare Logic

Abort

$\vdash \texttt{abort} \sim \texttt{abort} : \texttt{true} \Rightarrow \texttt{false}$

Rules of Relational Hoare Logic
Abort

$\vdash \texttt{abort}\sim\texttt{abort}:\texttt{true}\Rightarrow\texttt{false}$

To show this rule correct we need to show the validity of the quadruple $\texttt{abort}\sim\texttt{abort}:\texttt{T}\Rightarrow\texttt{F}$.

Rules of Relational Hoare Logic

Abort

$\vdash \texttt{abort} \sim \texttt{abort} : \texttt{true} \Rightarrow \texttt{false}$

To show this rule correct we need to show the validity of the quadruple $\texttt{abort} \sim \texttt{abort} : \texttt{T} \Rightarrow \texttt{F}$.

For every $\texttt{m}_1, \texttt{m}_2$ such that $\texttt{P}(\texttt{m}_1, \texttt{m}_2)$ we can show $\{\texttt{abort}\}_{\texttt{m1}} = \bot$ iff $\{\texttt{abort}\}_{\texttt{m2}} = \bot$.

Rules of Relational Hoare Logic
Abort

$\vdash \texttt{abort} \sim \texttt{abort} : \texttt{true} \Rightarrow \texttt{false}$

To show this rule correct we need to show the validity of the quadruple $\texttt{abort} \sim \texttt{abort} : T \Rightarrow F$.

For every m_1, m_2 such that $P(m_1, m_2)$ we can show $\{\texttt{abort}\}_{m1} = \bot$ iff $\{\texttt{abort}\}_{m2} = \bot$.

Follow easily by our semantics:

$\{\texttt{abort}\}_{m} = \bot$

Rules of Relational Hoare Logic

Assignment

$$\frac{}{\vdash x_1 := e_1 \sim x_2 := e_2 : P[e_1<1>/x_1<1>, e_2<2>/x_2<2>] \Rightarrow P}$$

Rules of Relational Hoare Logic

Assignment Example

$$\frac{}{\vdash \texttt{x:=x+1} \sim \texttt{y:=y-1}: \texttt{x<1>+1=-(y<2>-1)} \Rightarrow \texttt{x<1>=-y<2>}}$$

Rules of Relational Hoare Logic
Assignment Example

```
⊢x:=x+1 ~ y:=y-1:
 (x<1> = -y<2>)
 [(x<1>+1)/x<1>,(y<2>-1)/y<2>]
  ⇒
x<1> = -y<2>
```

Rules of Relational Hoare Logic
Consequence

$$\frac{P \Rightarrow S \qquad \vdash c_1 \sim c_2 : S \Rightarrow R \qquad R \Rightarrow Q}{\vdash c_1 \sim c_2 : P \Rightarrow Q}$$

We can weaken P, i.e. replace it by something that is implied by P. In this case S.

We can strengthen Q, i.e. replace it by something that implies Q. In this case R.

Consequence + Assignment

Example

$$\frac{}{\vdash \mathtt{x{:=}x{+}1} \sim \mathtt{y{:=}y{-}1} : \; \mathtt{x}\langle 1\rangle = -\mathtt{y}\langle 2\rangle \Rightarrow \mathtt{x}\langle 1\rangle = -\mathtt{y}\langle 2\rangle}$$

Consequence + Assignment

Example

$$\frac{\begin{array}{l} x\langle 1\rangle=-y\langle 2\rangle \Rightarrow x\langle 1\rangle+1=-(y\langle 2\rangle-1) \\ \vdash x:=x+1 \sim y:=y-1 : \\ x\langle 1\rangle+1=-(y\langle 2\rangle-1) \Rightarrow x\langle 1\rangle=-y\langle 2\rangle \\ x\langle 1\rangle=-y\langle 2\rangle \Rightarrow x\langle 1\rangle=-y\langle 2\rangle \end{array}}{\begin{array}{l} \vdash x:=x+1 \sim y:=y-1 : \\ x\langle 1\rangle=-y\langle 2\rangle \Rightarrow x\langle 1\rangle=-y\langle 2\rangle \end{array}}$$

Rules of Relational Hoare Logic Composition

$$\frac{\vdash c_1 \sim c_2 : P \Rightarrow R \qquad \vdash c_1' \sim c_2' : R \Rightarrow S}{\vdash c_1 ; c_1' \sim c_2 ; c_2' : P \Rightarrow S}$$

Rules of Hoare Logic
If then else

$$\dfrac{\begin{array}{c}\vdash c_1 \sim c_2 : e_1\langle 1\rangle \wedge e_2\langle 2\rangle \wedge P \Rightarrow Q \\ \vdash c_1' \sim c_2' : \neg e_1\langle 1\rangle \wedge \neg e_2\langle 2\rangle \wedge P \Rightarrow Q\end{array}}{\vdash \begin{array}{c}\texttt{if } e_1 \texttt{ then } c_1 \texttt{ else } c_1' \\ \sim \\ \texttt{if } e_2 \texttt{ then } c_2 \texttt{ else } c_2'\end{array} : P \Rightarrow Q}$$

Rules of Hoare Logic
If then else

$$\vdash c_1 \sim c_2 : e_1\langle 1\rangle \land e_2\langle 2\rangle \land P \Rightarrow Q$$

$$\vdash c_1' \sim c_2' : \neg e_1\langle 1\rangle \land \neg e_2\langle 2\rangle \land P \Rightarrow Q$$

$$\vdash \begin{array}{c} \texttt{if } e_1 \texttt{ then } c_1 \texttt{ else } c_1' \\ \sim \\ \texttt{if } e_2 \texttt{ then } c_2 \texttt{ else } c_2' \end{array} : P \Rightarrow Q$$

Is this correct?

An example

$$\vdash \begin{array}{c} \texttt{if true then skip else x:=x+1} \\ \sim \\ \texttt{if false then x:=x+1 else skip} \end{array} : \{\texttt{x<1>=n}\} \Rightarrow \{\texttt{x<1>=n+1}\}$$

Is this a valid quadruple?

An example

$$\vdash \begin{array}{c} \texttt{if true then skip else x:=x+1} \\ \sim \\ \texttt{if false then x:=x+1 else skip} \end{array} : \{x<1>=n\} \Rightarrow \{x<1>=n+1\}$$

Is this a valid quadruple? ✗

An example

$$\vdash \begin{array}{c} \texttt{if true then skip else x:=x+1} \\ \sim \\ \texttt{if false then x:=x+1 else skip} \end{array} : \{x\langle 1\rangle=n\} \Rightarrow \{x\langle 1\rangle=n+1\}$$

Is this a valid quadruple? ✗

Can we prove it with the rule above?

An example

$$\vdash \begin{array}{c}\texttt{if true then skip else x:=x+1} \\ \sim \\ \texttt{if false then x:=x+1 else skip}\end{array} : \{\texttt{x<1>=n}\} \Rightarrow \{\texttt{x<1>=n+1}\}$$

Is this a valid quadruple? ✗

Can we prove it with the rule above? ✓

Rules of Relational Hoare Logic
If then else

$$\frac{\begin{array}{l} P \Rightarrow e_1\langle 1\rangle = e_2\langle 2\rangle \\ \vdash c_1 \sim c_2 : e_1\langle 1\rangle \wedge P \Rightarrow Q \\ \vdash c_1' \sim c_2' : \neg e_1\langle 1\rangle \wedge P \Rightarrow Q \end{array}}{\vdash \begin{array}{c} \texttt{if } e_1 \texttt{ then } c_1 \texttt{ else } c_1' \\ \sim \\ \texttt{if } e_2 \texttt{ then } c_2 \texttt{ else } c_2' \end{array} : P \Rightarrow Q}$$

Rules of Hoare Logic
While

$$\frac{P \Rightarrow e_1\langle 1\rangle = e2\langle 2\rangle \qquad \vdash c_1 \sim c_2 : e_1\langle 1\rangle \wedge P \Rightarrow P}{\vdash \begin{array}{c}\texttt{while } e_1 \texttt{ do } c_1 \\ \sim \\ \texttt{while } e_2 \texttt{ do } c_2\end{array} : P \Rightarrow P \wedge \neg e_1\langle 1\rangle}$$

Invariant

How can we prove this?

```
x:private
y:public

x:=y
```

$: \; =_{low} \Rightarrow \; =_{low}$

How can we prove this?

```
x:private
y:public

y:=x
```

$: \; =_{low} \Rightarrow \; =_{low}$

How can we prove this?

```
x:private
y:public

y:=x
```

$: \quad =_{low} \Rightarrow \quad =_{low}$

Can we prove it?

How can we prove this?

```
x:private
y:public


y:=x
y:=5


:  =low ⇒  =low
```

How can we prove this?

```
x:private
y:public

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

$: \; =_{low} \Rightarrow \; =_{low}$

How can we prove this?

```
x:private
y:public

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

$: \; =_{\mathtt{low}} \Rightarrow =_{\mathtt{low}}$

How can we prove this?

```
x:private
y:public

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

Can we prove it?

$: \; =_{low} \Rightarrow \; =_{low}$

Rules of Relational Hoare Logic
If then else

$$P \Rightarrow e_1\langle 1\rangle = e_2\langle 2\rangle$$

$$\vdash c_1 \sim c_2 : e_1\langle 1\rangle \wedge P \Rightarrow Q$$

$$\vdash c_1' \sim c_2' : \neg e_1\langle 1\rangle \wedge P \Rightarrow Q$$

$$\vdash \begin{array}{c} \texttt{if } e_1 \texttt{ then } c_1 \texttt{ else } c_1' \\ \sim \\ \texttt{if } e_2 \texttt{ then } c_2 \texttt{ else } c_2' \end{array} : P \Rightarrow Q$$

Rules of Relational Hoare Logic
If then else - left

$$\vdash c_1 \sim c_2 : e\langle 1\rangle \land P \Rightarrow Q$$

$$\vdash c_1' \sim c_2 : \neg e\langle 1\rangle \land P \Rightarrow Q$$

$$\vdash \begin{array}{c} \texttt{if } e \texttt{ then } c_1 \texttt{ else } c_1' \\ \sim \\ c_2 \end{array} : P \Rightarrow Q$$

How can we prove this?

```
x:private
y:public

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

$: \; =_{\mathtt{low}} \Rightarrow =_{\mathtt{low}}$

Rules of Relational Hoare-Logic

One-sided Rules

What do we do if our two programs have different forms? There are three pairs of *one-sided* rules.

Rules of Relational Hoare Logic

If-then-else — left

$$\vdash c_1 \sim c_2 : e\langle 1\rangle \wedge P \Rightarrow Q$$

$$\vdash c_1' \sim c_2 : \neg e\langle 1\rangle \wedge P \Rightarrow Q$$

$$\vdash \begin{array}{c} \texttt{if } e \texttt{ then } c_1 \texttt{ else } c_1' \\ \sim \\ c_2 \end{array} : P \Rightarrow Q$$

Rules of Relational Hoare Logic

If-then-else — right

$$\frac{\vdash c_1 \sim c_2 : e\langle 2\rangle \wedge P \Rightarrow Q \qquad \vdash c_1 \sim c_2' : \neg e\langle 2\rangle \wedge P \Rightarrow Q}{\vdash \begin{array}{c} c_1 \\ \sim \\ \texttt{if } e \texttt{ then } c_2 \texttt{ else } c_2' \end{array} : P \Rightarrow Q}$$

Rules of Relational Hoare Logic

Assignment — left

$$\frac{}{\vdash \texttt{x:=e} \sim \texttt{skip}:\ \texttt{P[e<1>/x<1>]} \Rightarrow \texttt{P}}$$

Rules of Relational Hoare Logic

Assignment — right

$$\dfrac{}{\vdash \texttt{skip} \sim \texttt{x:=e}: \texttt{P[e<2>/x<2>]} \Rightarrow \texttt{P}}$$

Also pair of one-sided rules for while — we'll ignore for now

Rules of Relational Hoare-Logic
Rules for Program Equivalence

RHL also has some rules allowing one to reason modulo program equivalence - we will not see there here.

How can we prove this?

```
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

: n>0 /\ =low ⇒ ¬(=low)
```

How can we prove this?

```
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 do
 if not(s1[i]=s2[i]) then
    r:=1
 i:=i+1

: n>0 /\ =low ⇒ =low
```

Soundness

If we can derive $\vdash c_1 \sim c_2 : P \Rightarrow Q$ through the rules of the logic, then the quadruple $c_1 \sim c_2 : P \Rightarrow Q$ is valid.

Relative Completeness

If a quadruple $c_1 \sim c_2 : P \Rightarrow Q$ is valid, and we have an oracle to derive all the true statements of the form $P \Rightarrow S$ and of the form $R \Rightarrow Q$, then we can derive $\vdash c_1 \sim c_2 : P \Rightarrow Q$ through the rules of the logic.

Soundness and completeness with respect to Hoare Logic

$$\vdash_{\mathtt{RHL}} \mathtt{c_1 \sim c_2 : P \Rightarrow Q}$$

iff

$$\vdash_{\mathtt{HL}} \mathtt{c_1 ; c_2 : P \Rightarrow Q}$$

Soundness and completeness with respect to Hoare Logic

$$\vdash_{\mathtt{RHL}} \mathtt{c_1 \sim c_2 : P \Rightarrow Q}$$

iff

$$\vdash_{\mathtt{HL}} \mathtt{c_1 ; c_2 : P \Rightarrow Q}$$

Under the assumption that we can partition the memory adequately, and that we have termination.

Probabilistic Language

An example

```
OneTimePad(m : private msg) : public msg
   key :=$ Uniform({0,1}^n);
   cipher := msg xor key;
   return cipher
```

Learning a ciphertext does not change any a priori knowledge about the likelihood of messages.

Probabilistic While (PWhile)

```
c::= abort
   | skip
   | x:= e
   | x:=$ d
   | c;c
   | if e then c else c
   | while e do c
```

$d_1, d_2, \ldots$ probabilistic expressions

Probabilistic Expressions

We extend the language with expression describing probability distributions.

$$\boxed{\texttt{d::= f}(e_1,\ldots,e_n,d_1,\ldots,d_k)}$$

Where `f` is a distribution declaration

Some expression examples

`uniform(`$\{0,1\}^n$`)`   `gaussian(`k,σ`)`   `laplace(k,b)`

Semantics of Probabilistic Expressions

We would like to define it on the structure:

$$\{f(e_1,\ldots,e_n,d_1,\ldots,d_k)\}_m = \{f\}(\{e_1\}_m,\ldots,\{e_n\}_m,\{d_1\}_m,\ldots,\{d_k\}_m)$$

but is the result just a value?

Probabilistic Subdistributions

A discrete subdistribution over a set A is a function
$\mu : A \to [0, 1]$
such that the mass of μ,

$$|\mu| = \sum_{a \in A} \mu(a)$$

verifies $|\mu| \leq 1$.

The support of a discrete subdistribution μ,
$\mathrm{supp}(\mu) = \{a \in A \mid \mu(a) > 0\}$
is necessarily countable, i.e. finite or countably infinite.

We will denote the set of sub-distributions over A by D(A),
and say that μ is of type D(A) denoted μ:D(A) if $\mu \in D(A)$.

Probabilistic Subdistributions

We call a subdistribution with mass exactly 1, a distribution.

We define the probability of an event $E \subseteq A$ with respect to the subdistribution $\mu : D(A)$ as

$$\mathbb{P}_\mu[E] = \sum_{a \in E} \mu(a)$$

Probabilistic Subdistributions

Let's consider μ∈D(A), and E⊆A, we have the following properties

$\mathbb{P}_\mu[\varnothing] = 0$

$\mathbb{P}_\mu[A] \leq 1$

$0 \leq \mathbb{P}_\mu[E] \leq 1$

E⊆F⊆A implies $\mathbb{P}_\mu[E] \leq \mathbb{P}_\mu[F]$

E⊆A and F⊆A implies $\mathbb{P}_\mu[E \cup F] \leq \mathbb{P}_\mu[E] + \mathbb{P}_\mu[F] - \mathbb{P}_\mu[E \cap F]$

We will denote by **O** the subdistribution μ defined as constant 0.

Operations over Probabilistic Subdistributions

Let's consider an arbitrary a∈A, we will often use the distribution unit(a) defined as:

$$\mathbb{P}_{\text{unit}(a)}[\{b\}] = \begin{cases} \text{1 if a=b} \\ \text{0 otherwise} \end{cases}$$

We can think about unit as a function of type unit:A $\rightarrow$ D(A)

Operations over Probabilistic Subdistributions

Let's consider a distribution $\mu \in D(A)$, and a function $M: A \rightarrow D(B)$ then we can define their composition by means of an expression let a =μ in M a defined as:

$$\mathbb{P}_{\text{let a =}\mu\text{ in M a}}[E] = \sum_{a \in \mathrm{supp}(\mu)} \mathbb{P}_{\mu}[\{a\}] \cdot \mathbb{P}_{(Ma)}[E]$$

Semantics of Probabilistic Expressions - revisited

We would like to define it on the structure:

$$\{f(e_1,\ldots,e_n,d_1,\ldots,d_k)\}_m = \{f\}(\{e_1\}_m,\ldots,\{e_n\}_m,\{d_1\}_m,\ldots,\{d_k\}_m)$$

With input a memory m and output a subdistribution μ∈D(A) over the corresponding type A. E.g.

$$\{\mathtt{uniform}(\{0,1\}^n)\}_m \in D(\{0,1\}^n)$$

$$\{\mathtt{gaussian}(k,\sigma)\}_m \in D(\mathtt{Real})$$

Semantics of PWhile Commands

What is the meaning of the following command?

```
k:=$ uniform({0,1}^n); z:= x mod k;
```

Semantics of PWhile Commands

What is the meaning of the following command?

```
k:=$ uniform({0,1}^n); z:= x mod k;
```

We can give the semantics as a function between command, memories and subdistributions over memories.

$$\texttt{Cmd * Mem} \longrightarrow \texttt{D(Mem)}$$

We will denote this relation as:

$$\{c\}_m = \mu$$

Semantics of Commands

This is defined on the structure of commands:

Semantics of Commands

This is defined on the structure of commands:

$$\{\texttt{abort}\}_{\texttt{m}} = \mathbf{0}$$

Semantics of Commands

This is defined on the structure of commands:

$$\{\texttt{abort}\}_m = \mathbf{0}$$

$$\{\texttt{skip}\}_m = \texttt{unit(m)}$$

Semantics of Commands

This is defined on the structure of commands:

$$\{\texttt{abort}\}_m = \mathbf{0}$$

$$\{\texttt{skip}\}_m = \texttt{unit}(m)$$

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

Semantics of Commands

This is defined on the structure of commands:

$\{\texttt{abort}\}_{\texttt{m}} = \mathbf{0}$

$\{\texttt{skip}\}_{\texttt{m}} = \texttt{unit(m)}$

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

$\{\texttt{c;c}'\}_{\texttt{m}} = \texttt{let m}' = \{\texttt{c}\}_{\texttt{m}} \texttt{ in } \{\texttt{c}'\}_{\texttt{m}'}$

Semantics of Commands

This is defined on the structure of commands:

$\{\texttt{abort}\}_m = \mathbf{0}$

$\{\texttt{skip}\}_m = \texttt{unit}(m)$

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

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

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

Semantics of Commands

This is defined on the structure of commands:

$\{\texttt{abort}\}_{m} = \mathbf{0}$

$\{\texttt{skip}\}_{m} = \texttt{unit(m)}$

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

$\{\texttt{c;c'}\}_{m} = \texttt{let m'=}\{\texttt{c}\}_{m}\texttt{ in }\{\texttt{c'}\}_{m'}$

$\{\texttt{if e then c}_{t}\texttt{ else c}_{f}\}_{m} = \{\texttt{c}_{t}\}_{m}$ If $\{\texttt{e}\}_{m}\texttt{=true}$

$\{\texttt{if e then c}_{t}\texttt{ else c}_{f}\}_{m} = \{\texttt{c}_{f}\}_{m}$ If $\{\texttt{e}\}_{m}\texttt{=false}$

Semantics of Commands

This is defined on the structure of commands:

$\{\texttt{abort}\}_m = \mathbf{0}$

$\{\texttt{skip}\}_m = \texttt{unit(m)}$

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

$\{\texttt{x:=\$ d}\}_m = \texttt{let a=}\{\texttt{d}\}_m \texttt{ in unit(m[x}\leftarrow\texttt{a])}$

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

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

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

Semantics of While

What about while

How did we handle the deterministic case?

Semantics of While

What about while

$$\{\texttt{while e do c}\}_{\texttt{m}} = \texttt{???}$$

How did we handle the deterministic case?

Semantics of While

We defined it as

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

Where

$\mu_n =$

$\texttt{let m'=\{(while}^n\texttt{ e do c)\}}_m\texttt{ in \{if e then abort\}}_{m'}$

Semantics of While

We defined it as

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

Where

$$\mu_n =$$

$$\texttt{let } m' = \{(\texttt{while}^n \texttt{ e do c})\}_m \texttt{ in } \{\texttt{if e then abort}\}_{m'}$$

Is this well defined?

Semantics of Commands

This is defined on the structure of commands:

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

$\mu_n = \texttt{let } m' = \{(\texttt{while}^n \texttt{ e do c})\}_m \texttt{ in } \{\texttt{if e then abort}\}_{m'}$

Semantics of Commands

This is defined on the structure of commands:

$\{\texttt{abort}\}_m = \mathbf{0}$

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

$\mu_n = \texttt{let } m' = \{(\texttt{while}^n \texttt{ e do c})\}_m \texttt{ in } \{\texttt{if e then abort}\}_{m'}$

Semantics of Commands

This is defined on the structure of commands:

$\{\texttt{abort}\}_m = \mathbf{0}$

$\{\texttt{skip}\}_m = \texttt{unit}(m)$

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

$\mu_n = \texttt{let } m' = \{(\texttt{while}^n \texttt{ e do c})\}_m \texttt{ in } \{\texttt{if e then abort}\}_{m'}$

Semantics of Commands

This is defined on the structure of commands:

$\{\texttt{abort}\}_m = \mathbf{0}$

$\{\texttt{skip}\}_m = \texttt{unit}(m)$

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

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

$\mu_n = \texttt{let } m' = \{(\texttt{while}^n\ e \texttt{ do } c)\}_m \texttt{ in } \{\texttt{if } e \texttt{ then abort}\}_{m'}$

Semantics of Commands

This is defined on the structure of commands:

$\{\texttt{abort}\}_m = \mathbf{0}$

$\{\texttt{skip}\}_m = \texttt{unit}(m)$

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

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

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

$\mu_n = \texttt{let } m' = \{(\texttt{while}^n \texttt{ e do c})\}_m \texttt{ in } \{\texttt{if e then abort}\}_{m'}$

Semantics of Commands

This is defined on the structure of commands:

$\{\texttt{abort}\}_m = \mathbf{0}$

$\{\texttt{skip}\}_m = \texttt{unit}(m)$

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

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

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

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

$\mu_n = \texttt{let } m' = \{(\texttt{while}^n \texttt{ e do c})\}_m \texttt{ in } \{\texttt{if e then abort}\}_{m'}$

Semantics of Commands

This is defined on the structure of commands:

$\{\texttt{abort}\}_m = \mathbf{0}$

$\{\texttt{skip}\}_m = \texttt{unit}(m)$

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

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

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

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

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

$\mu_n = \texttt{let } m' = \{(\texttt{while}^n \ e \texttt{ do } c)\}_m \texttt{ in } \{\texttt{if } e \texttt{ then abort}\}_{m'}$

Semantics of Commands

This is defined on the structure of commands:

$\{\texttt{abort}\}_m = \mathbf{0}$

$\{\texttt{skip}\}_m = \texttt{unit}(m)$

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

$\{x :=\$\ d\}_m = \texttt{let } a = \{d\}_m \texttt{ in unit}(m[x \leftarrow a])$

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

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

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

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

$\mu_n = \texttt{let } m' = \{(\texttt{while}^n\ e \texttt{ do } c)\}_m \texttt{ in } \{\texttt{if } e \texttt{ then abort}\}_{m'}$

Revisiting the example

```
OneTimePad(m : private msg) : public msg
   key :=$ Uniform({0,1}^n);
   cipher := msg xor key;
   return cipher
```

Learning a ciphertext does not change any a priori knowledge about the likelihood of messages.

Revisiting the example

```
OneTimePad(m : private msg) : public msg
   key :=$ Uniform({0,1}^n);
   cipher := msg xor key;
   return cipher
```

Learning a ciphertext does not change any a priori knowledge about the likelihood of messages.

How do we formalize this?

Probabilistic Noninterference

A program `prog` is probabilistically noninterferent if and only if, whenever we run it on two low equivalent memories m_1 and m_2 we have that the probabilistic distributions we get as outputs are the same on public outputs.