CS 591: Formal Methods in Security and Privacy
Noninterference and Relational Hoare Logic

Marco Gaboardi
gaboardi@bu.edu

Alley Stoughton
stough@bu.edu
From the previous classes
Information Flow Control

We want to guarantee that confidential inputs do not flow to nonconfidential outputs.
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: $m_1 \sim_{\text{low}} m_2$
Noninterference

A program $\text{prog}$ 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 $m_1 \sim_{\text{low}} m_2$:
1) $\{c\}_m = \bot$ iff $\{c\}_m = \bot$
2) $\{c\}_m = m_1'$ and $\{c\}_m = m_2'$ implies $m_1' \sim_{\text{low}} m_2'$
Does this program satisfy noninterference?

\[
\begin{array}{c}
x: \text{private} \\
y: \text{public} \\
x := y
\end{array}
\]

Yes
Does this program satisfy noninterference?

\[
\begin{array}{l}
x: \text{private} \\
y: \text{public} \\
y := x
\end{array}
\]

No
Is this program secure?

\[
\begin{align*}
x & : \text{private} \\
y & : \text{public} \\
y & := x \\
y & := 5
\end{align*}
\]

Yes
Does this program satisfy noninterference?

\[ \text{x:private} \]
\[ \text{y:public} \]

\[
\text{if } x \mod 3 = 0 \text{ then} \\
\quad \text{y:=1} \\
\text{else} \\
\quad \text{y:=0}
\]

No - an “implicit flow”
How can we prove our programs noninterferent?
Relational Property

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

1) \( \{c\}_{m_1} = \bot \) iff \( \{c\}_{m_2} = \bot \)

2) \( \{c\}_{m_1} = m_1' \) and \( \{c\}_{m_2} = m_2' \) implies \( m_1' \sim_{\text{low}} m_2' \)
Relational Hoare Logic - RHL

Precondition

Program$_1 \sim$ Program$_2$

Postcondition

$c_1 \sim c_2 : P \Rightarrow Q$

Precondition (a logical formula)

Program

Program

Postcondition (a logical formula)
Relational Assertions

\[ c_1 \sim c_2 : P \implies Q \]

Need to talk about variables of the two memories

\[ c_1 \sim c_2 : x\langle 1 \rangle \leq x\langle 2 \rangle \implies x\langle 1 \rangle \geq x\langle 2 \rangle \]

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) $\{c_1\}_{m_1} = \perp$ iff $\{c_2\}_{m_2} = \perp$

2) $\{c_1\}_{m_1} = m_1'$ and $\{c_2\}_{m_2} = m_2'$ implies $Q(m_1', m_2')$.

How do we check this?
Rules of Relational Hoare Logic
Skip

\[ \vdash \text{skip} \leadsto \text{skip}: \top \Rightarrow \top \]
Abort

\[ \neg \text{abort} \sim \neg \text{abort} : \text{true} \Rightarrow \text{false} \]
Rules of Relational Hoare Logic
Assignment

⊢ \( x_1 := e_1 \sim x_2 := e_2 : \)
\( P [ e_1 <1> / x_1 <1> ,
    e_2 <2> / x_2 <2> ] \) ⇒
\( P \)

What is changed from last class?
Today: More Relational Hoare Logic
Rules of Relational Hoare Logic
Assignment Example

\[ \Gamma \vdash x := x + 1 \sim y := y - 1 : \\
(x_{>1} = -y_{<2}>)
\]
\[ [(x+1)_{<1}>/x_{<1>}, \\
(y-1)_{<2}>/y_{<2}>] \Rightarrow \\
x_{<1>} = -y_{<2}> \]
Rules of Relational Hoare Logic
Assignment Example

\[ \vdash x := x + 1 \sim y := y - 1 : \]
\[ (x < 1 > = -y < 2 >) \]
\[ \left[ \frac{(x < 1 > + 1)}{x < 1 >} , (y < 2 > - 1)}{y < 2 >} \right] \Rightarrow x < 1 > = -y < 2 > \]
Rules of Relational Hoare Logic
Assignment example

\[ \Gamma x := x + 1 \sim y := y - 1 : \]
\[ x^{<1>} + 1 = -(y^{<2>} - 1) \Rightarrow \]
\[ x^{<1>} = -y^{<2>} \]
Rules of Relational Hoare Logic Composition

\[ \vdash c_1 \sim c_2 : P \Rightarrow R \quad \vdash c_1' \sim c_2' : R \Rightarrow S \]

\[ \vdash c_1 ; c_1' \sim c_2 ; c_2' : P \Rightarrow S \]
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

\[ x<1> = -y<2> \Rightarrow x<1> + 1 = -(y<2> - 1) \]

\[ \vdash x := x + 1 \sim y := y - 1: \\
    x<1> + 1 = -(y<2> - 1) \Rightarrow x<1> = -y<2> \]

\[ x<1> = -y<2> \Rightarrow x<1> = -y<2> \]

\[ \vdash x := x + 1 \sim y := y - 1: \\
    x<1> = -y<2> \Rightarrow x<1> = -y<2> \]
Rules of Relational Hoare Logic

If-then-else

\[ \vdash c_1 \sim c_2 : e_1 <1> \land e_2 <2> \land P \Rightarrow Q \]

\[ \vdash c_1' \sim c_2' : \neg e_1 <1> \land \neg e_2 <2> \land P \Rightarrow Q \]

\[ \vdash \sim : P \Rightarrow Q \]

Is this correct?
Rules of Relational Hoare Logic
If-then-else

\[ P \Rightarrow (e_1 <1> \iff e_2 <2>) \]

\( \vdash c_1 \sim c_2 : e_1 <1> \land P \Rightarrow Q \)

\( \vdash c_1' \sim c_2' : \neg e_1 <1> \land P \Rightarrow Q \)

\[ \vdash \sim : P \Rightarrow Q \]

if \( e_1 \) then \( c_1 \) else \( c_1' \)

if \( e_2 \) then \( c_2 \) else \( c_2' \)
Rules of Relational Hoare Logic

While

\[ P \implies (e_1 \triangleleft 1 \iff e_2 \triangleleft 2) \]

\[ \vdash c_1 \sim c_2 : e_1 \triangleleft 1 \land P \implies P \]

\[ \vdash \sim : P \implies P \land \neg e_1 \triangleleft 1 \]

Invariant
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.

\[
\text{if } e \text{ then } c_1 \text{ else } c_1' \quad \vdash \quad \sim \quad \vdash \quad \because P \Rightarrow Q
\]

\[
c_2
\]
Rules of Relational Hoare Logic
If-then-else — left

\[ \vdash c_1 \sim c_2 : e < 1 > \land P \Rightarrow Q \]
\[ \vdash c_1' \sim c_2 : \neg e < 1 > \land P \Rightarrow Q \]

if e then \( c_1 \) else \( c_1' \)
\[ \vdash \sim c_2 : P \Rightarrow Q \]
Rules of Relational Hoare Logic
If-then-else — right

\[ \vdash c_1 \sim c_2 : e<2> \land P \Rightarrow Q \]
\[ \vdash c_1 \sim c_2' : \lnot e<2> \land P \Rightarrow Q \]

\[ \vdash \sim \begin{array}{c}
\text{if } e \text{ then } c_2 \text{ else } c_2' \\
\end{array} : P \Rightarrow Q \]
Rules of Relational Hoare Logic
Assignment — left

\[\vdash x := e \sim \text{skip}: \quad \mathcal{P}[e_{<1>}/x_{<1>}] \Rightarrow \mathcal{P}\]
Rules of Relational Hoare Logic
Assignment — right

\[ \vdash \text{skip} ~ \sim ~ x := e : \]
\[ P[e<2>/x<2>] \Rightarrow P \]

Also pair of one-sided rules for while — we’ll ignore for now
Program Equivalence Rule

\[ \models P : c_1 \equiv c_2 \text{ means } \{ c_1 \}_m = \{ c_2 \}_m \text{ for all } m \text{ such that } P(m) \]

\[ \models P : c_1' \equiv c_1 \quad \models P : c_2' \equiv c_2 \]

\[ c_1' \sim c_2' : P \Rightarrow Q \]

\[ \models c_1 \sim c_2 : P \Rightarrow Q \]
Rules of Relational Hoare Logic
Program Equivalences

⊢P : skip;c ≡ c

⊢P : c;skip ≡ c

⊢P : (c1;c2);c3 ≡ c1;(c2;c3)

...
We can combine the Composition and Program Equivalence Rules to split commands where we like:

\[
\vdash c_1; c_2 \sim c_1' : P \Rightarrow R
\]

\[
\vdash c_3 \sim c_2' ; c_3' : R \Rightarrow Q
\]

\[
\vdash c_1; c_2; c_3 \sim c_1' ; c_2' ; c_3' : P \Rightarrow Q
\]
Rules of Relational Hoare Logic
Combining Composition and Equivalence

\[ \vdash \begin{align*}
  c_1 \sim \text{skip} : P & \Rightarrow R \\
  c_2 \sim c_1' : R & \Rightarrow Q \\
\end{align*} \]

\[ \vdash \begin{align*}
  c_1 ; c_2 \sim \text{skip} ; c_1' : P & \Rightarrow Q \\
\end{align*} \]
Rules of Relational Hoare Logic
Combining Composition and Equivalence

\[ \vdash c_1 \sim c_1': P \Rightarrow R \]

\[ \vdash c_2 \sim \text{skip}: R \Rightarrow Q \]

\[ \vdash c_1; c_2 \sim c_1'; \text{skip}: P \Rightarrow Q \]

\[ \vdash c_1; c_2 \sim c_1': P \Rightarrow Q \]
Relational Hoare Logic in EasyCrypt

• EasyCrypt’s implementation of Relational Hoare Logic has much in common with its implementation of Hoare Logic.

• Look for the pRHL tactics in Section 3.4 of the EasyCrypt Reference Manual (the “p” stands for “probabilistic”, but ignore that for now).
In Lab next, we’ll look at some noninterference proofs in EasyCrypt