Static analysis (of high-level languages)

Ana Matos
Miguel Correia Pedro Adão

☞ "Secure information flow: definition, enforcement, and preservation through compilation", G. Barthe, B. Grégoire, A. Matos, T. Rezk, 2011

Topics

- Steps for Information flow analyses
  - Noninterference
  - Type systems for noninterference

Steps for an IFlow analysis

1. Definition of the language
2. Information flow policy of security levels
3. Classification of objects into security levels
4. Security property of programs
5. Mechanism for selecting secure programs
6. Guarantees about the mechanism

To formalize a language

- Define its syntax.
  - Ex: Using BNF notation
- Define its semantics.
  - Ex: Using a big-step transition system

- Today: We will use the WHILE language

Syntax of WHILE

- Syntactic categories:
  $c \in Z$ - constants (integers)
  $x \in Var$ - variables
  $a \in Aexp$ - arithmetic expressions
  $t \in Bexp$ - tests
  $S \in Stm$ - statements

- Grammar (BNF notation):

  op ::= + | - | × | /   cmp ::= < | ≤ | = | ≠ | ≥ | >

  a ::= c | x | a_1 op a_2 t ::= a_1 cmp a_2

  S ::= x:=a | skip | S_1 ; S_2 | if t then S else S | while t do S

Big-step semantics of WHILE

Assignment: $< x{:=}a, \rho > \rightarrow \rho[x \mapsto \mathcal{A}[\![a]\!]_\rho]$

Skip: $< \mathrm{skip}, \rho > \rightarrow \rho$

Sequential composition: $$\frac{<S_1, \rho> \rightarrow \rho' \quad <\rho_2, \rho'> \rightarrow \rho''}{< S_1;S_2, \rho> \rightarrow \rho''}$$

Conditional test: $$\frac{<S_1, \rho> \rightarrow \rho'}{<\text{if } t \text{ then } S_1 \text{ else } S_2, \rho> \rightarrow \rho'} \quad \text{if } \mathcal{B}[\![t]\!]_\rho = \text{true}$$

$$\frac{<S_2, \rho> \rightarrow \rho'}{<\text{if } t \text{ then } S_1 \text{ else } S_2, \rho> \rightarrow \rho'} \quad \text{if } \mathcal{B}[\![t]\!]_\rho = \text{false}$$

While loop: $$\frac{<S,\rho> \rightarrow \rho' \quad <\text{while } t \text{ do } S, \rho'> \rightarrow \rho''}{<\text{while } t \text{ do } S, \rho> \rightarrow \rho''} \quad \text{if } \mathcal{B}[\![t]\!]_\rho = \text{true}$$

$$<\text{while } t \text{ do } S, \rho> \rightarrow \rho \qquad \text{if } \mathcal{B}[\![t]\!]_\rho = \text{false}$$

Steps for an IFlow analysis

1. A WHILE language, with a natural operational semantics.
2. Information flow policy of security levels
3. Classification of objects into security levels
4. Security property of programs
5. Mechanism for selecting secure programs
6. Guarantees about the mechanism

2. To define a policy of security levels

- We choose a set of security classes, an flow relation between them, and an operator for combining them. These security levels can speak for instance of confidentiality, integrity, or both.
- Today: We will use a lattice of security levels

Lattice policies

- A class of common information flow policies have some convenient ingredients:
  - Security levels form a partial order (allowed flows are transitive, anti-symmetric a).
  - Two security levels can always be combined.
  - There is a highest and a lowest level.

Lattice policies

- When an flow relation has a lattice structure:

$$\mathcal{L} = (L, \leq, \vee, \wedge, \top, \bot)$$

Set of security levels

Partial order on L

Join: Least upper bound operation

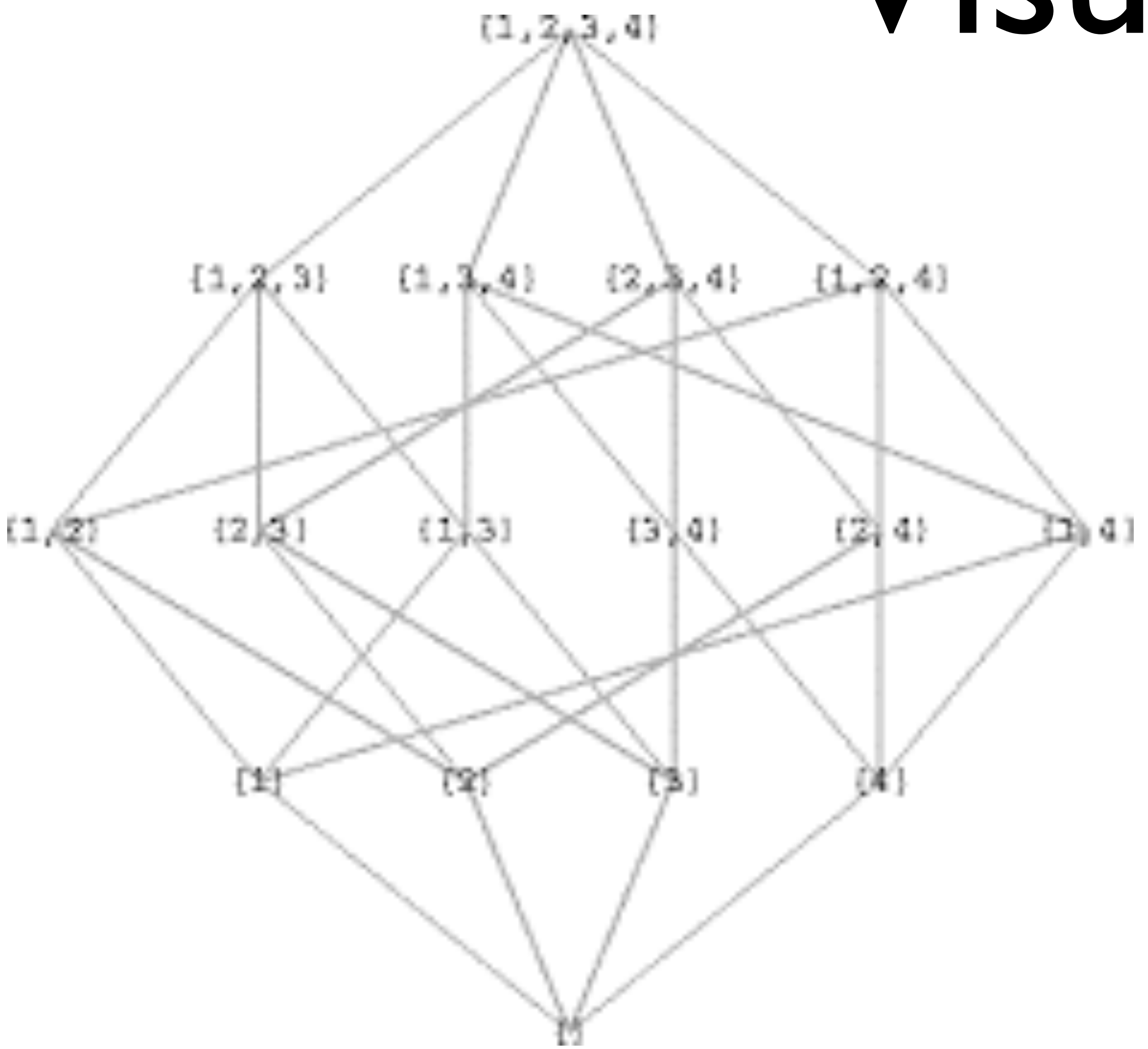
Meet: Greatest lower bound operation

Top security level

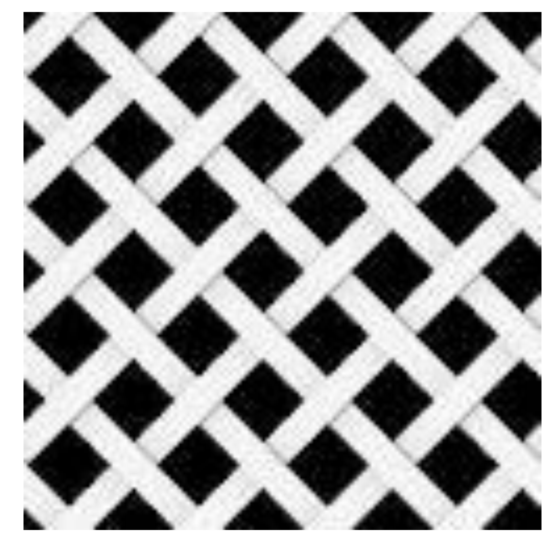
Bottom security level

- Operation $\oplus$ is then given by $\vee$ or $\wedge$

Visually

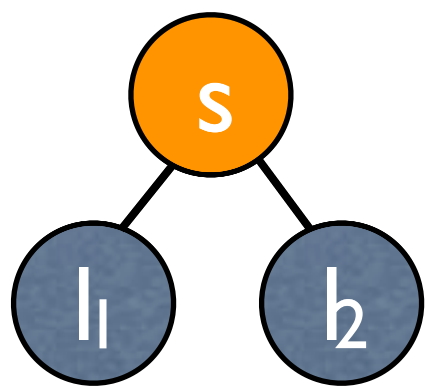


Lattice of subsets of the set
{1,2,3,4},
ordered by the subset relation.

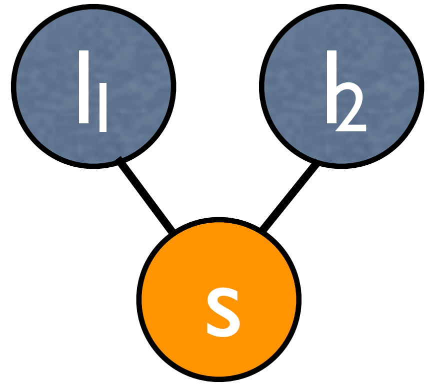


Upper and lower bounds

- Upper-Bound - Given a partially ordered set $(S, \leq)$, element s is an upper-bound of two elements l_1 and l_2 if it satisfies:
  - $l_1 \leq s$ and $l_2 \leq s$

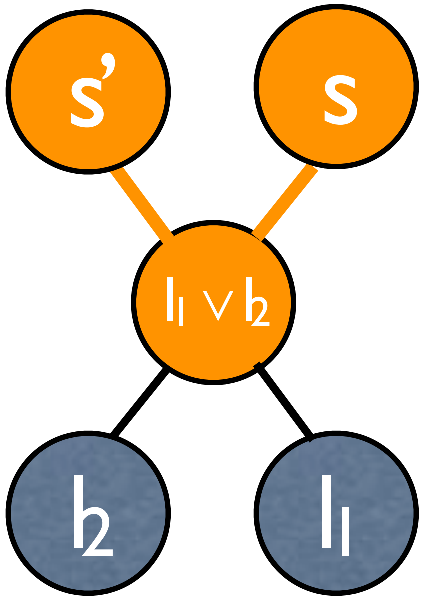


- Lower-Bound - Given a partially ordered set $(S, \leq)$, element s is a lower-bound of two elements l_1 and l_2 if it satisfies
  - $s \leq l_1$ and $s \leq l_2$

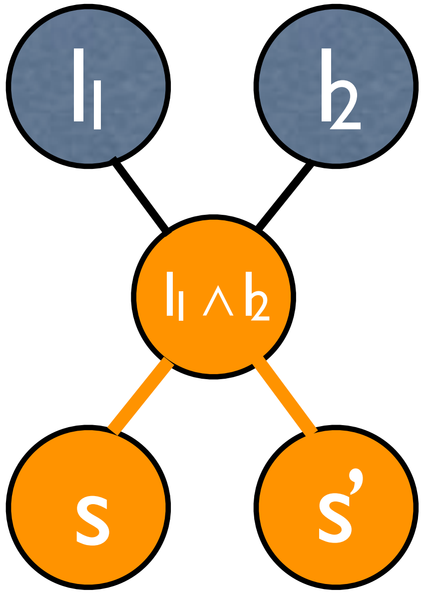


Join and Meet

- Join ($\vee$) - defined for any two $l_1, l_2 \in L$, we have that $l_1 \vee l_2$ is the least upper-bound of l_1 and l_2:
  - all other upper-bounds s of l_1 and l_2 are greater: $l_1 \vee l_2 \leq s$



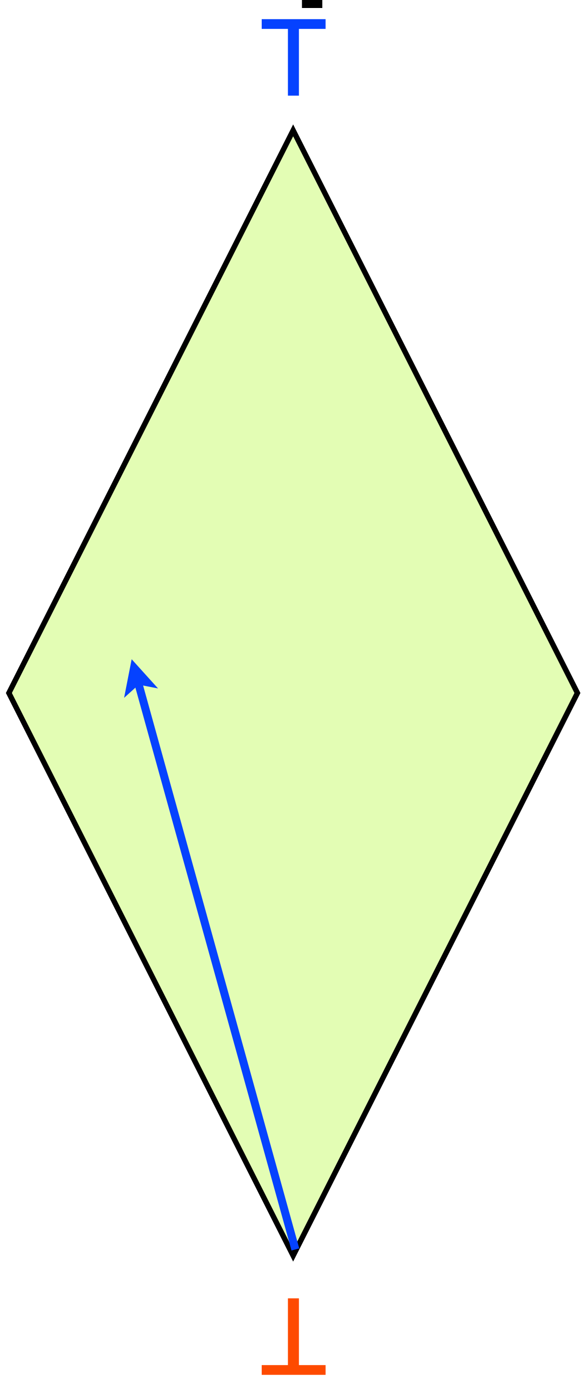
- Meet ($\wedge$) - defined for any two $l_1, l_2 \in L$, we have that $l_1 \wedge l_2$ is the greatest lower-bound of l_1 and l_2.
  - all other lower-bounds s of l_1 and l_2 are lower: $s \leq l_1 \wedge l_2$



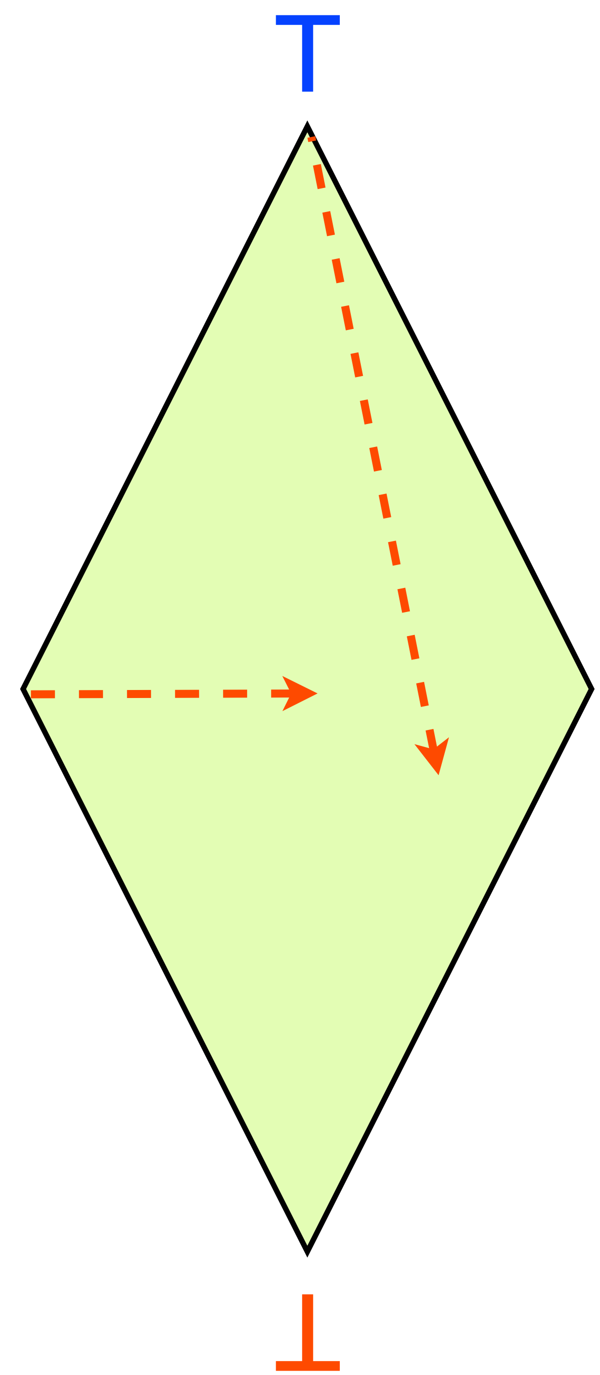
Which are lattices?

- Isolated classes policy.
- High-Low policy for confidentiality.
- High-Low policy for integrity.
- Principal-based policy for confidentiality.
- Principal-based policy for integrity.

Information Flow policies as lattices



Allowed flows: upwards!



Lattice of confidentiality levels

Set of security levels

Can-flow relation (partial order)

Top: highest confidentiality level

- $\mathcal{L} = (L, \leq, \vee, \wedge, \top, \bot)$

Join: least upper bound operation

Meet: greatest lower bound operation

Bottom: lowest confidentiality level

Steps for an IFlow analysis

1. A WHILE language, with a natural operational semantics.
2. A lattice L of confidentiality levels.
3. Classification of objects into security levels
4. Security property of programs
5. Mechanism for selecting secure programs
6. Guarantees about the mechanism

3. Objects to security levels

- Objects, or information containers (variables, channels, files, etc) are given a security level.

- Today: We will use a security labeling, i.e. a mapping from variables to security levels ($\Gamma : \mathit{Var} \rightarrow L$)

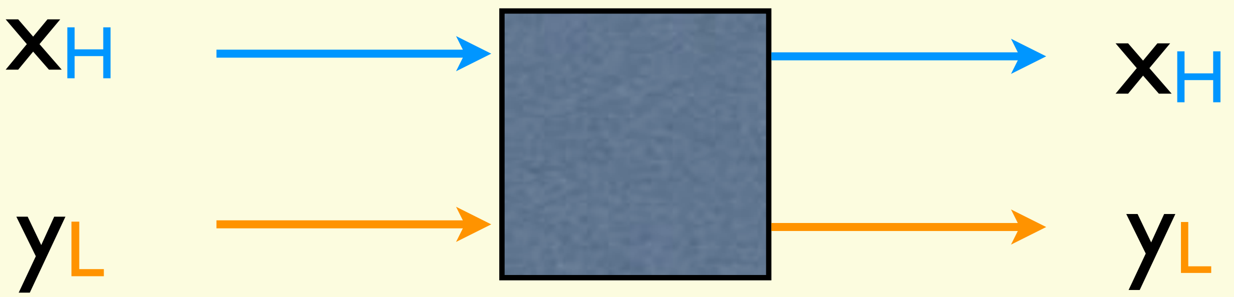
Steps for an IFlow analysis

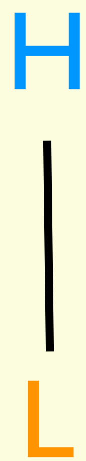
1. A WHILE language, with a structural operational semantics.
2. A lattice L of confidentiality levels.
3. Objects are variables that are given security levels by Γ.
4. Security property of programs
5. Mechanism for selecting secure programs
6. Guarantees about the mechanism.

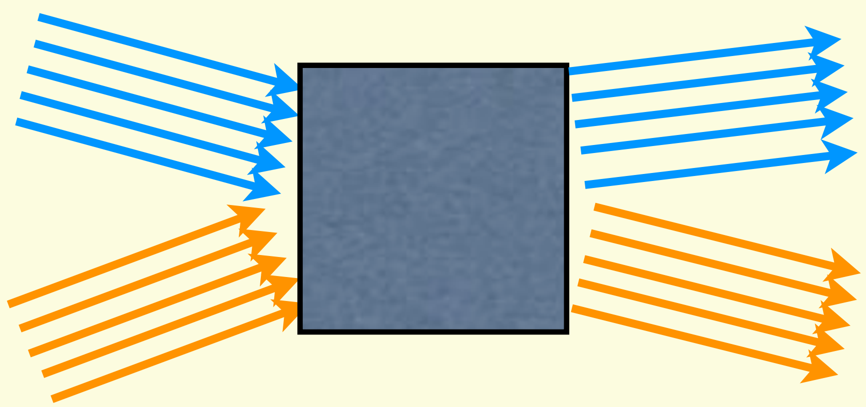
4. Formal security property of programs

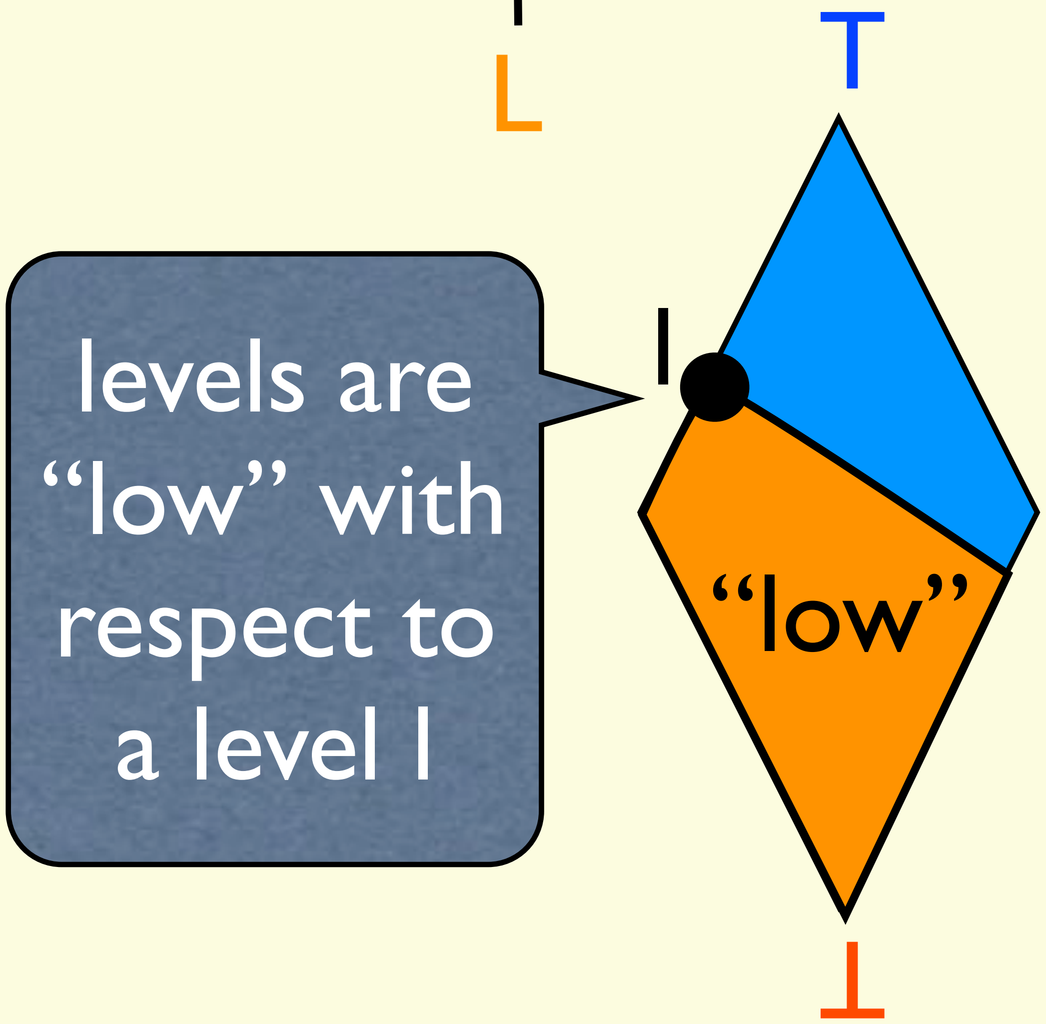
- While taking into consideration the language expressivity and execution context, specify the power of the attacker.
- Formal specification of what it means for a program to be secure.
- Today: We will use Deterministic Input-Output Noninterference

Noninterference, more generally



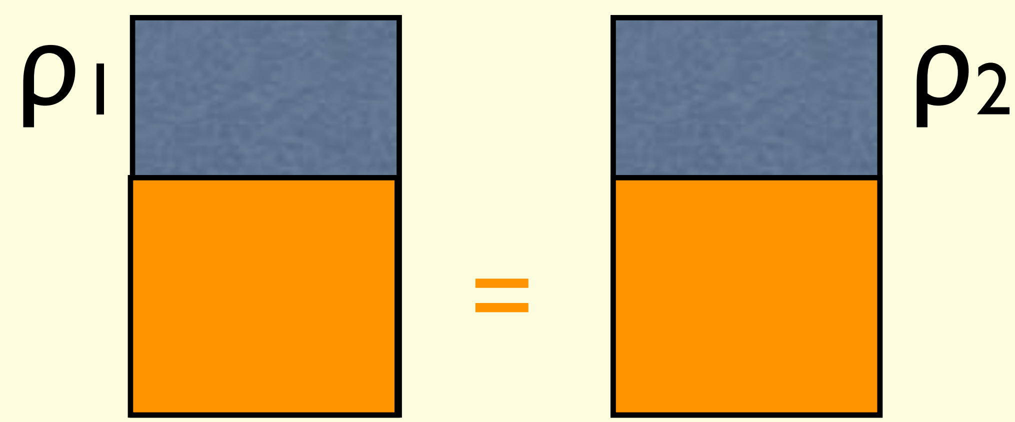






Indistinguishability

- Two memories ρ_1 and ρ_2 are indistinguishable with respect to a security labelling Γ and a level l, written $\rho_1 \sim_l^{\Gamma} \rho_2$, if ρ_1 and ρ_2 agree on the values of variables that are lower than l. I.e.: For all variables x such that $\Gamma(x) \leq l$, $\rho_1(x) = \rho_2(x)$.



- When Γ is clear from context, we omit it and simply write $\rho_1 \sim_l \rho_2$.

Deterministic Input-Output Noninterference

- **Deterministic Input-Output Noninterference** -
A program S is secure if for every security level L and for all pairs of memories ρ_1 and ρ_2 such that $\rho_1 \sim_L \rho_2$, we have that
$\langle S, \rho_1 \rangle \rightarrow \rho_1'$ and $\langle S, \rho_2 \rangle \rightarrow \rho_2'$ implies $\rho_1' \sim_L \rho_2'$.

Steps for an IFlow analysis

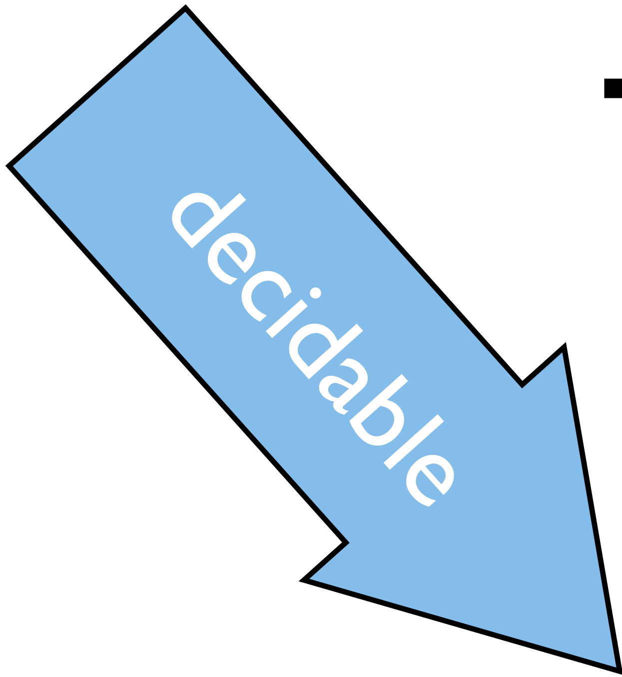
1. A WHILE language, with a structural operational semantics.
2. A lattice L of confidentiality levels.
3. Objects are variables that are given security levels by Γ.
4. Input-output Noninterference.
5. Mechanism for selecting secure programs
6. Guarantees about the mechanism.

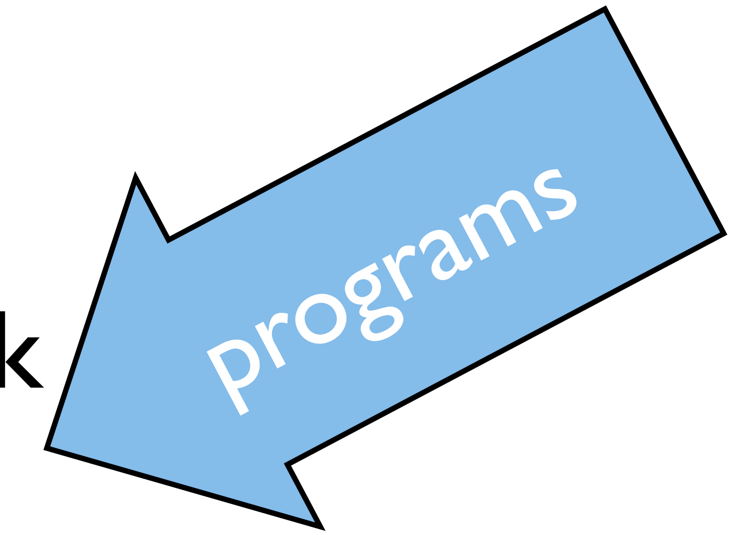
Building a type system

Challenge for next class

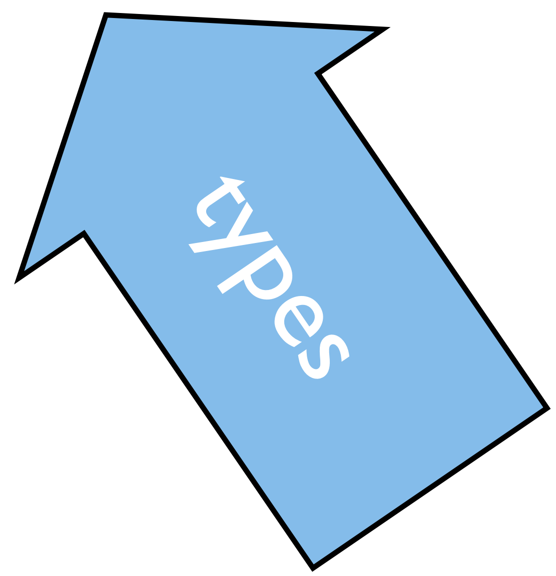
- Can you propose an algorithm for statically selecting secure programs of our WHILE language?
- Tips:
    - Purely *syntactic* properties are decidable. So, focus on the possible syntax of programs.
    - For each possible syntactic construction of a program, can you think of rules of thumb for deciding roughly whether a program should be accepted or not?

Type systems





- "Tractable syntactic framework
for classifying phrases
according to the kinds of values they
compute"



-- B. Pierce, 2002

Aim of the type system

Our type system should:

- analyze programs written in our WHILE language.
- only accept programs that satisfy Noninterference.

Type system - ingredients

- **Types**
- **Typing environment Γ**
  maps variables to types.
- **Judgments Γ⊢program:type**
  partially map programs to types.
- **Rules** $\dfrac{\textbf{assumptions}}{\textbf{conclusion}}$
  relate valid judgments with assumptions

to know what are the types of the variables

for stating that a (part of a) program is typable and what is its type

for establishing syntactic conditions for judgments to hold

Idea

- Chose a level that represents those of the variables that are read in an expression or test
- Chose a level that represents those of the variables that are written in a statement
- Only type statements where written variables only depend on read variables (expressions or test) of lower or equal level

Type system for Noninterference - idea

- **For typing expressions**
  - Types: τ
  - Judgments: $\Gamma \vdash a : \tau$

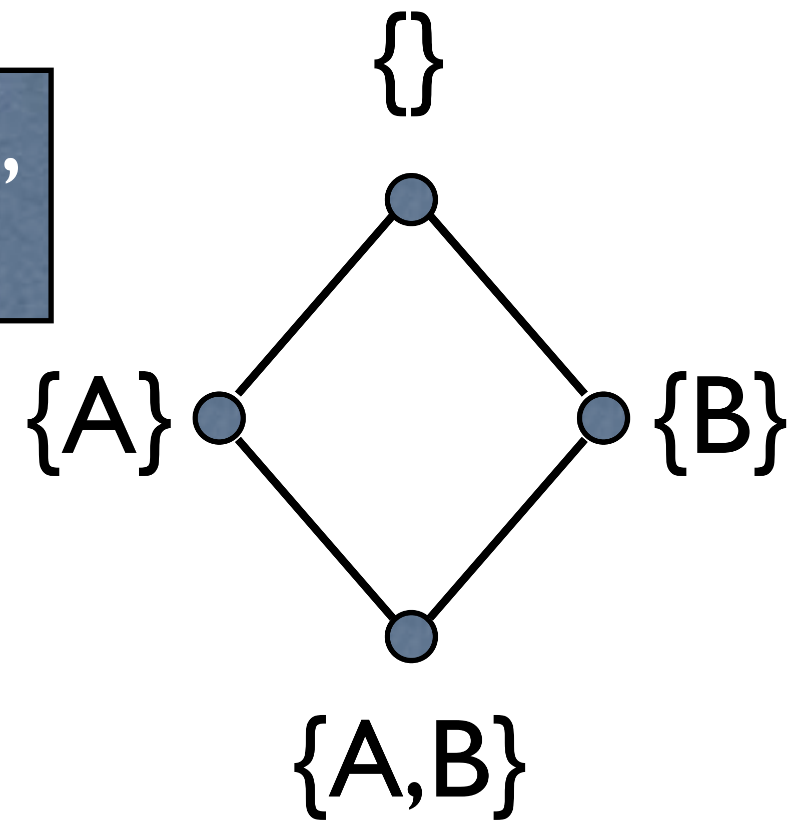
upper bound to read variables

- **For typing statements (commands)**
  - Types: τ cmd
  - Judgments: $\Gamma \vdash S : \tau$ cmd

lower bound to written variables

Types to expressions

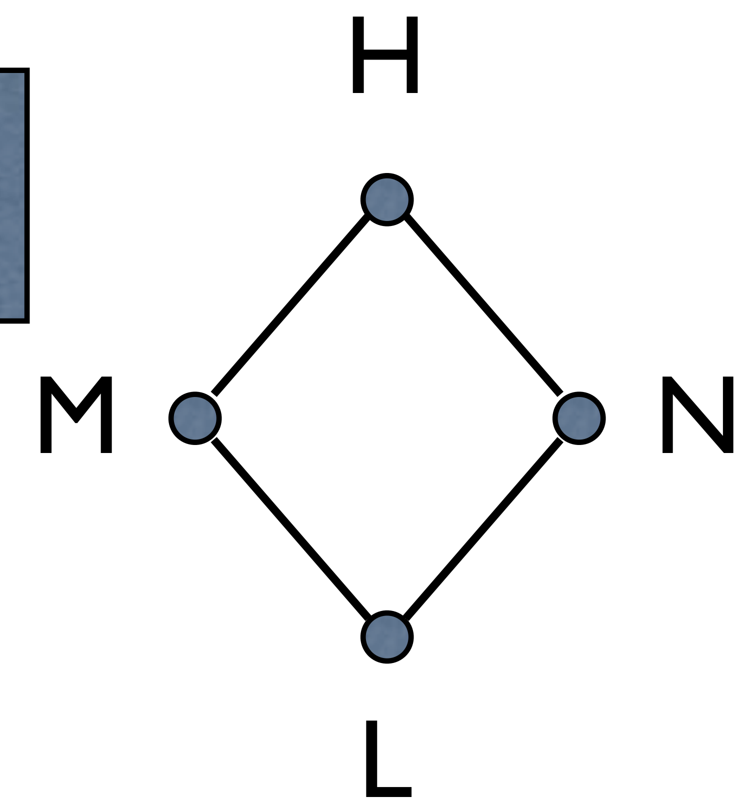
"upper bound to variables that are read"



- What could be their types?
  - $(x_{\{A\}} + y_{\{B\}}) * z_{\{A,B\}}$ {}
  - $x_{\{A,B\}} < (y_{\{A,B\}} + 1)$ {A,B}, {A}, {B}, {}
  - $y_{\{B\}} == z_{\{B\}} - x_{\{A,B\}}$ {B}, {A,B}

Types to expressions

> "upper bound to variables that are read"

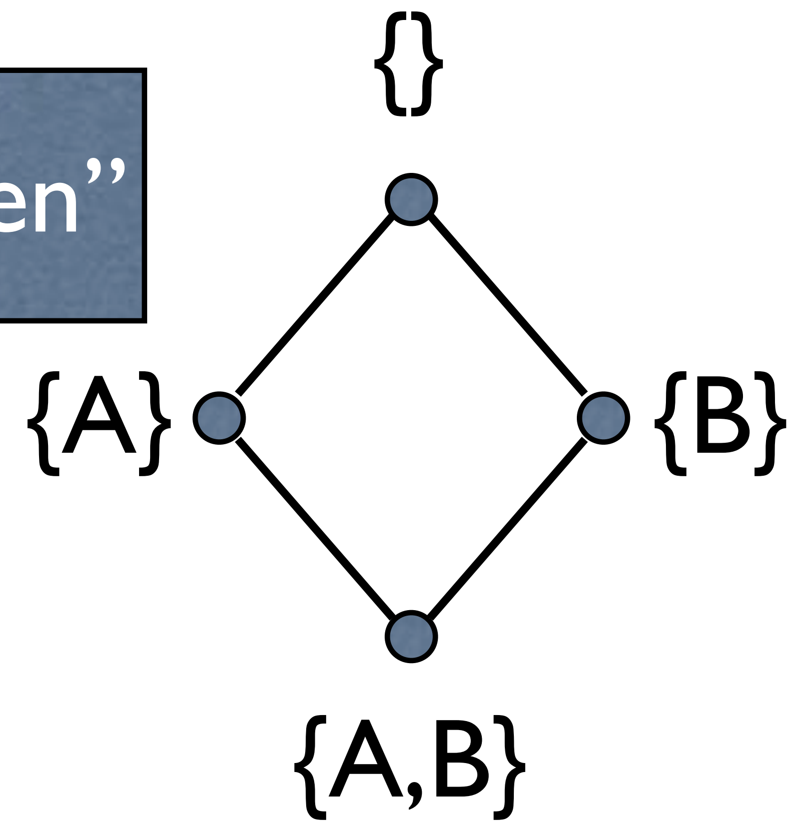


- What could be their types?
  - $(x_M + y_N) * z_L$ H
  - $x_L < (y_L + 1)$ L, M, N, H
  - $y_M = z_M - x_L$ M,H
- How about…
  - xH - xH
  - zM = zM

Can you think of more complicated ways of encoding constants?

Types to commands

"lower bound to variables that are written"



- What could be their types?
  - $y_{\{B\}} := (x_{\{A\}} + y_{\{B\}}) * z_{\{A,B\}}$
  - if($x_{\{A,B\}} < (y_{\{A,B\}} + 1)$) then $z_{\{A\}} := 1$ else $w_{\{B\}} := 1$

    {A,B} cmd
  - $y_{\{\}} := z_{\{B\}} - x_{\{A,B\}}$; $y_{\{\}} := y_{\{\}} - x_{\{A,B\}}$

    {} cmd, {A} cmd, {B} cmd, {A,B} cmd

Types of expressions are at least as high as that of its sub-expressions (remaining an upper bound!).

- Constants:

$$\Gamma \vdash c : \bot$$

Has no read variables.

Gets lowest level of all.

Types of expressions are at least as high as that of its sub-expressions (remaining an upper bound!).

- Variables: $\dfrac{\Gamma(x) = \tau}{\Gamma \vdash x : \tau}$

Only one read variable

Gets the level of that variable (lowest possible)

Types of expressions are at least as high as that of its sub-expressions (remaining an upper bound!).

Least upper bounds of each expression

Gets the least upper bounds of both

- Expressions:

$$\frac{\Gamma \vdash a_1 : \tau_1 \quad \Gamma \vdash a_2 : \tau_2}{\Gamma \vdash a_1 \ \mathrm{op}\ a_2 : \tau_1 \vee \tau_2} \qquad \frac{\Gamma \vdash a_1 : \tau_1 \quad \Gamma \vdash a_2 : \tau_2}{\Gamma \vdash a_1 \ \mathrm{cmp}\ a_2 : \tau_1 \vee \tau_2}$$

Typing rules for expressions

- Constants:

$$\Gamma \vdash c : \bot$$

- Variables: $$\frac{\Gamma(x) = \tau}{\Gamma \vdash x : \tau}$$

- Expressions:

$$\frac{\Gamma \vdash a_1 : \tau_1 \quad \Gamma \vdash a_2 : \tau_2}{\Gamma \vdash a_1 \text{ op } a_2 : \tau_1 \vee \tau_2} \qquad \frac{\Gamma \vdash a_1 : \tau_1 \quad \Gamma \vdash a_2 : \tau_2}{\Gamma \vdash a_1 \text{ cmp } a_2 : \tau_1 \vee \tau_2}$$

Types of statements are at least as low as those of its sub-statements (remaining a lower bound!).

- Skip: $\Gamma \vdash \text{skip} : \top \text{ cmd}$

Has no written variables.

Gets highest level of all.

Types of statements are at least as low as those of its sub-statements (remaining a lower bound!).

Reject low assignments of high expressions and low writes under high guards.

Read level is at least as low as written variable

- Assignment:

$$\frac{\Gamma(x) = \tau \qquad \Gamma \vdash a : \tau' \qquad \tau' \leq \tau}{\Gamma \vdash x := a : \tau \text{ cmd}}$$

Has one written variable.

Gets the level of that variable

Types of statements are at least as low as those of its sub-statements (remaining a lower bound!).

Greatest lower bounds of each expression

Gets the greatest bounds of both

- Sequential composition:

$$\frac{\Gamma \vdash S_1 : \tau_1 \text{ cmd} \qquad \Gamma \vdash S_2 : \tau_2 \text{ cmd}}{\Gamma \vdash S_1;S_2 : \tau_1 \wedge \tau_2 \text{ cmd}}$$

Types of statements are at least as low as those of its sub-statements (remaining a lower bound!).

Reject low assignments of high expressions and low writes under high guards.

- Conditional test:

$$\frac{\Gamma \vdash t : \tau \quad \Gamma \vdash S_1 : \tau_1 \text{ cmd} \quad \Gamma \vdash S_2 : \tau_2 \text{ cmd} \quad \tau \leq \tau_1, \tau_2}{\Gamma \vdash \text{if } t \text{ then } S_1 \text{ else } S_2 : \tau_1 \wedge \tau_2 \text{ cmd}}$$

Tested level is at least as low as variables potentially written in branches

Types of statements are at least as low as those of its sub-statements (remaining a lower bound!).

Reject low assignments of high expressions and low writes under high guards.

Tested level is at least as low as variables potentially written in branches

- While loop:

$$\frac{\Gamma \vdash t : \tau \quad \Gamma \vdash S : \tau' \text{ cmd} \quad \tau \leq \tau'}{\Gamma \vdash \text{while } t \text{ do } S : \tau' \text{ cmd}}$$

Typing rules for statements

- Skip: $\Gamma \vdash \text{skip} : \top \text{ cmd}$
- Assignment:
$$\frac{\Gamma(x) = \tau \quad \Gamma \vdash a : \tau' \quad \tau' \leq \tau}{\Gamma \vdash x := a : \tau \text{ cmd}}$$
- Sequential composition:
$$\frac{\Gamma \vdash S_1 : \tau_1 \text{ cmd} \qquad \Gamma \vdash S_2 : \tau_2 \text{ cmd}}{\Gamma \vdash S_1;S_2 : \tau_1 \wedge \tau_2 \text{ cmd}}$$
- Conditional test:
$$\frac{\Gamma \vdash t : \tau \quad \Gamma \vdash S_1 : \tau_1 \text{ cmd} \quad \Gamma \vdash S_2 : \tau_2 \text{ cmd} \quad \tau \leq \tau_1, \tau_2}{\Gamma \vdash \text{if } t \text{ then } S_1 \text{ else } S_2 : \tau_1 \wedge \tau_2 \text{ cmd}}$$
- While loop:
$$\frac{\Gamma \vdash t : \tau \quad \Gamma \vdash S : \tau' \text{ cmd} \quad \tau \leq \tau'}{\Gamma \vdash \text{while } t \text{ do } S : \tau' \text{ cmd}}$$

Typing derivation tree

- Try to derive a type for statement
  $y := z \,;\, x := 42$ when $\Gamma(x)=L$, $\Gamma(y)=H$ and $\Gamma(z)=L$.

$\tau_3 = L$ $\tau_4 = \perp$

$\tau_1 = H$ $\tau_2 = L$

$$\dfrac{\dfrac{\Gamma(y)=H \qquad \dfrac{\Gamma(z)=L}{\Gamma \vdash z : \tau_3} \qquad \tau_3 \leq H}{\Gamma \vdash\ y := z : \tau_1 \text{ cmd}} \qquad \dfrac{\Gamma(x)=L \qquad \Gamma \vdash 42 : \tau_4 \qquad \tau_4 \leq L}{\Gamma \vdash\ x := 42 : \tau_2 \text{ cmd}}}{\Gamma \vdash\ y := z \,;\, x := 42 : \tau_1 \wedge \tau_2 \text{ cmd}}$$

Typing derivation tree

- Try to derive a type for statement
  if y=1 then x := 0 else x := 1 when $\Gamma(x)=L$ and $\Gamma(y)=H$.

How precise is our type system?

- $x_L := y_H$ ✗
- $y_H := z_H \; ; x_L := w_L$ ✓
- if $y_H = 1$ then $x_L := 0$ else $x_L := 1$ ✗
- if $y_H = 1$ then $x_L := 0$ else $x_L := 0$ ✗
- while $y_H = 1$ do skip ; $x_L := 0$ ✓

This secure program is rejected (as expected!)

Steps for an IFlow analysis

1. A WHILE language, with a structural operational semantics.
2. A lattice L of confidentiality levels.
3. Objects are variables that are given security levels by Γ.
4. Input-output Noninterference.
5. Information flow type system.
6. Guarantees about the mechanism.

Lemma 1
Simple Security

Is τ indeed an upper bound to read variables?

- Simple Security:
  If $\Gamma \vdash a : \tau$ then expression a contains only variables of level τ or lower.
- Reasoning:
  Types given to expressions are obtained from types of its variables, and can only be raised.

Lemma 2
Confinement

Is τ indeed a lower bound to written variables?

- Confinement:
  If $\Gamma \vdash S : \tau$ cmd then statement S assigns only to variables of level τ or higher.
- Reasoning:
  Types given to statements are obtained from types of its assigned variables, and can only be lowered.

Theorem
Type Soundness

Does the type system really enforce the property?

- Type soundness (informally):
  If program S is typable then S satisfies noninterference.

Theorem
Type Soundness

Does the type system really enforce the property?

- Type soundness (formally):
  If $\Gamma \vdash S : \tau$ cmd, then
  for every security level l and memories ρ_l and ρ_2
  such that $\rho_l \sim_l \rho_2$, we have that
  $<S, \rho_l> \longrightarrow \rho_l'$ and $<S, \rho_2> \longrightarrow \rho_2'$ implies $\rho_l' \sim_l \rho_2'$.

Steps for an IFlow analysis

1. A WHILE language, with a structural operational semantics.
2. A lattice L of confidentiality levels.
3. Objects are variables that are given security levels by Γ.
4. Input-output Noninterference.
5. Information flow type system.
6. Proof that the type system only accepts secure programs.

Conclusions

- We have developed a **static type-based analysis** for our WHILE language.
- It is possible to **prove** that this mechanism is **sound** with respect to Deterministic Input-Output Noninterference
- The same can be done for **other** languages, other variants of Noninterference and even other security properties (see next classes).