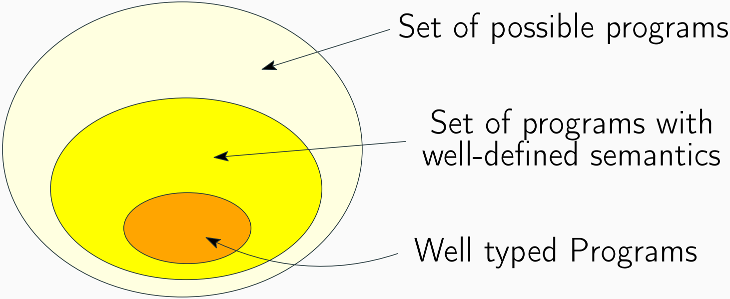
CSCI 740 - Programming Language Theory

Lecture 18
Types for Information Flow
Instructor: Hossein Hojjat

October 13, 2017

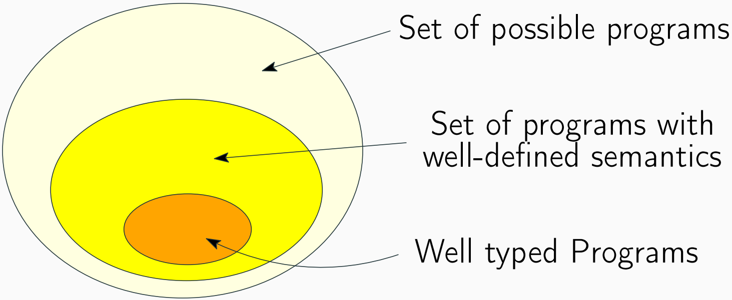
Recap



Functional World

- Evaluation proceeds through reduction rules
- Types impose constraints on the shape of the program
- A program with a legal shape (according to a correct type system):
    - Always has an available reduction rule (unless it has terminated)
    - Reduction rule will produce a new program with a legal shape

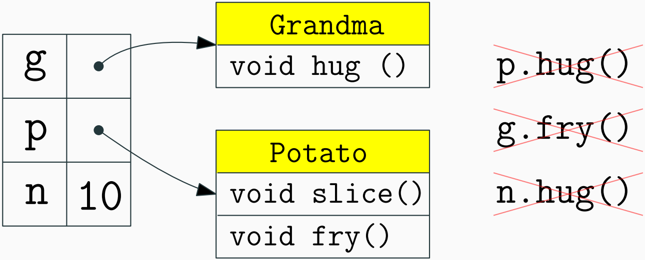
Recap



Imperative World

- Evaluation involves updating a store
- Types place restrictions on the program store
- This allows static reasoning about legal operations on the objects in the store

Recap



Imperative World

- Evaluation involves updating a store
- Types place restrictions on the program store
- This allows static reasoning about legal operations on the objects in the store

Motivation: Information Leakage

- Explicit flow

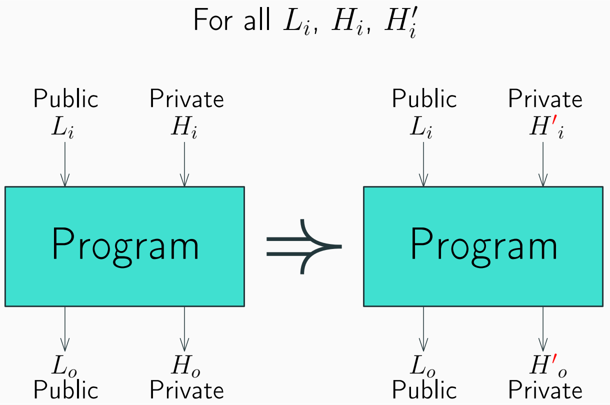
```
int secret;
...
int pub = secret;
```

- Implicit flow

```
boolean secret;
...
int pub = 0;
if(secret) pub = 1;
```

What is information flow?

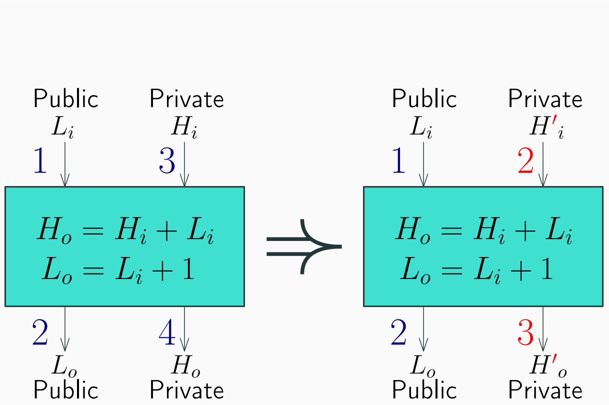
- If there is no information flow from private to public, then a change in a private input can not affect a public output
  - Impossible to verify this property by a single execution



"Commands of high-security users have no effect on observations of low-security users"

What is information flow?

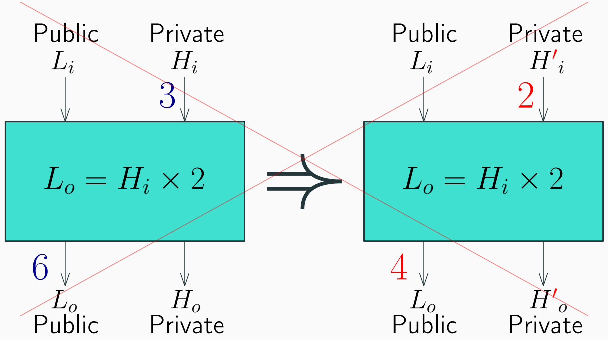
- If there is no information flow from private to public, then a change in a private input can not affect a public output
  - Impossible to verify this property by a single execution



"Commands of high-security users have no effect on observations of low-security users"

What is information flow?

- If there is no information flow from private to public, then a change in a private input can not affect a public output
  - Impossible to verify this property by a single execution



"Commands of high-security users have no effect on observations of low-security users"

JFlow: Type Annotations

- JFlow works by adding policies as type annotations
- Idea: attach security labels to types to obtain security types

```
int{secret} x;
int{public} y;
x = y; // ok if public ⊑ secret
y = x; // not ok if secret ⋢ public
```

Solution Strategy

We proceed through the following two steps:

Dynamic Semantics

- Define a labeling scheme so that at any given time, the labels of data tell if it's OK to leak it or not
- Labels turn a global property about all executions into a local property in a conservative way
- This will be the dynamic semantics against which we can prove type safety

Static Semantics

- Define a type system that allows us to approximate the set of labels that the data pointed at by a variable can have.
- If an action is OK according to the conservative approximation, we know it would be OK according to the dynamic scheme

Labeling Data With Security Policies

Policies for information flow

```
Owner:  reader1, reader2, reader3
```

- According to owner, this data can only be read by reader1, reader2, or reader3

Label

```
{ policy1, policy2, policy3 }
```

- If an owner is not mentioned, it is assumed she has no privacy concerns
- $L = \{o_1 :\}$ $\quad$ o_1 allows only his/herself to read
- $\{\}$ is the least-restrictive / most-public label
  - (no owner has expressed an interest in restarting the data)

Principals

- Owners and readers are principals
  - user, group or role
- `act_for` relationship
  - Allows principals to act for other principals

Hossein `act_for` Faculty

Labels form a Lattice

$$\boldsymbol{L_1 \sqsubseteq L_2}$$

- L_1 can be relabeled to L_2
- Means that L_2 is more restrictive (fewer readers)
- Warning: this is counterintuitive: L_2 actually has fewer readers
- If a variable is certified to handle data with L_2 labels correctly, we can trust that variable to hold a value with label L_1
  - Just like subtyping!

Labels form a Lattice

- Relation $\sqsubseteq$ should be: reflexive, transitive, and antisymmetric
- Labels form lattices
- Let $\ell_1 \sqcup \ell_2$ be join/LUB of ℓ_1 and ℓ_2

- Example lattice where only principals are Alice and Bob

{Alice: , Bob: }

{Alice: } = {Alice: , Bob: Alice }

{Bob: } = {Alice: Bob , Bob: }

{ } = {Alice: Bob , Bob: Alice }

Labels form a lattice

Question:

$$\{\text{Joe: Ann, Jill}\} \sqsubseteq \{\text{Joe: Ann}\}$$

$$\{\text{Joe: (Ann, Jill), Tim: Ann}\} \sqsubseteq \{\text{Joe: (Ann), Tim: Ann}\}$$

$$\{\text{Joe: (Ann), Tim: Ann}\} ??? \{\text{Joe: (Ann)}\}$$

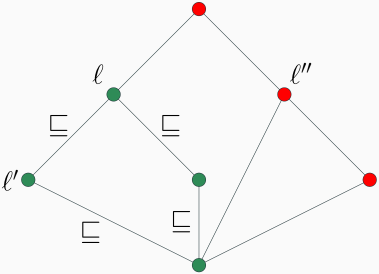
Labels form a lattice

Question:

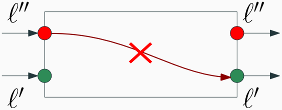
$$\{\text{Joe: Ann, Jill}\} \sqsubseteq \{\text{Joe: Ann}\}$$
$$\{\text{Joe: (Ann, Jill), Tim: Ann}\} \sqsubseteq \{\text{Joe: (Ann), Tim: Ann}\}$$
$$\{\text{Joe: (Ann), Tim: Ann}\} ??? \{\text{Joe: (Ann)}\}$$

- Notice that {Joe:(Ann)} is equivalent to {Joe:(Ann), Tim: Joe}
- So these two elements are incomparable

Labels form a Lattice



- Green labels are considered "low" with respect to ℓ
- Red labels are considered "high" with respect to ℓ
  - (either more restrictive or incomparable)
- Values tagged with red labels should not flow to values tagged with green labels



Assignment

$$x\{\ell_2\} := v\{\ell_1\};$$

$$\ell_1 \sqsubseteq \ell_2$$

- Can only assign to a variable to a more restrictive label

Binary Operations

$$a\{\ell_1\} + b\{\ell_2\};$$

Trick question:

- What should be the label for a+b?
- What information would be leaked if this code were to execute?

```
int{Joe:everyone} a, b, c;
...
int{Joe:Joe} p;
c = 0;
if(p){
    c = a + b;
}
```

Typing Rule for Binray Expressions

$$\frac{\Gamma \vdash e : \ell \qquad \Gamma \vdash e' : \ell'}{\Gamma \vdash e + e' : \ell \sqcup \ell'}$$

- $\sqcup$: least upper bound (lub)
- Most precise element that is a conservative approximation of both labels