

# Information Flow Analysis in Logical Form

Torben Amtoft & Anindya Banerjee  
Kansas State University

SAS in Verona, August 26, 2004

## Information Flow Analysis (a reminder)

**Confidentiality** the attacker cannot learn about **initial** value of **high** variable  $h$  from **final** value of **low** variable  $l$

$h := l$	secure
$l := 7$	secure
$l := h$	<b>insecure</b> ( <b>direct</b> flow)
if $h$ then $l := 7$ else $l := 8$	<b>insecure</b> ( <b>indirect</b> flow)
$l := h; l := 7$	secure (though insecure parts)

## Information Flow Analysis (a reminder)

**Confidentiality** the attacker cannot learn about **initial** value of **high** variable  $h$  from **final** value of **low** variable  $l$

$h := l$	secure
$l := 7$	secure
$l := h$	<b>insecure</b> ( <b>direct</b> flow)
if $h$ then $l := 7$ else $l := 8$	<b>insecure</b> ( <b>indirect</b> flow)
$l := h; l := 7$	secure (though insecure parts)

**Equivalent**: the final value of a low variable is **independent** of the initial value of a high variable

## Information Flow Analysis (a reminder)

**Confidentiality** the attacker cannot learn about **initial** value of **high** variable  $h$  from **final** value of **low** variable  $l$

$h := l$	secure
$l := 7$	secure
$l := h$	<b>insecure</b> ( <b>direct</b> flow)
if $h$ then $l := 7$ else $l := 8$	<b>insecure</b> ( <b>indirect</b> flow)
$l := h; l := 7$	secure (though insecure parts)

**Equivalent**: the final value of a **low** variable is **independent** of the initial value of a **high** variable

**Integrity**: the final value of a **licensed** variable is **independent** of the initial value of a **hacked** variable

## Contributions

- ▶ captures notion of variable independence in logical form
- ▶ allows efficient computation of invariants (using “strongest postcondition” function)
- ▶ shows our logical form subsumes classical type-based approach
- ▶ allows modular reasoning (frame rule)
- ▶ facilitates the generation of “counterexamples”

## Traces

**Language:** Simple imperative with `while`

A trace  $t \in \mathbf{Trc}$

- ▶ associates to each variable its **initial** and its **current** value.  
Example:  $t_1$  may be given as  $[x \mapsto (0, 3), y \mapsto (5, 2)]$ .
- ▶ is thus isomorphic to a pair of stores  
(records only endpoints of path)

The **semantics** models commands as mappings from sets of traces to sets of traces:

$$\llbracket \cdot \rrbracket : \mathcal{P}(\mathbf{Trc}) \rightarrow \mathcal{P}(\mathbf{Trc})$$

Example:  $\llbracket x := x + 1 \rrbracket$  maps  $\{t_1\}$  to  $\{[x \mapsto (0, 4), y \mapsto (5, 2)]\}$

## Trace Semantics

The semantics maps each **input trace** to **one or zero output traces**

Example:  $\llbracket \text{while } x > 7 \text{ do } x := x + 1 \rrbracket$  maps  
 $\{[x \mapsto (6, 6)], [x \mapsto (8, 8)]\}$  to  $\{[x \mapsto (6, 6)]\}$

**Semantics** ( $T \in \mathcal{P}(\mathbf{Trc})$ )

$$\llbracket x := E \rrbracket = \lambda T. \{t' \mid \exists t \in T : t' = t[x \mapsto \llbracket E \rrbracket(t)]\}$$

$$\llbracket C_1 ; C_2 \rrbracket = \lambda T. \llbracket C_2 \rrbracket(\llbracket C_1 \rrbracket(T))$$

$$\llbracket \text{if } E \text{ then } C_1 \text{ else } C_2 \rrbracket = \lambda T. \llbracket C_1 \rrbracket(E\text{-true}(T)) \cup \llbracket C_2 \rrbracket(E\text{-false}(T))$$

$$\llbracket \text{while } E \text{ do } C_0 \rrbracket = \text{Ifp}(\mathcal{F}) \text{ where } \mathcal{F}(f) = \lambda T. \dots$$

## Motivation

abstract

$T_0^\#$

$\models$

concrete

$T_0$

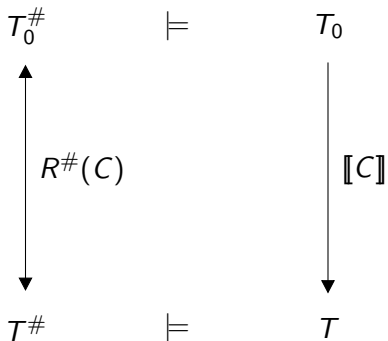
$\downarrow$   
[C]

$T$

## Motivation

abstract

concrete



## Independences

- ▶  $[y \# w]$  denotes that the **current** value of  $y$  **must** be independent of the **initial** value of  $w$
- ▶  $T^\#$  ranges over set of **independences**

$T^\# \models T$  holds iff for all  $[y \# w] \in T^\#, t_1, t_2 \in T$ :

- ▶ if  $t_1$  and  $t_2$  agree on the **initial** value of all variables **except**  $w$
- ▶ then  $t_1$  and  $t_2$  **agree** on the **final** value of  $y$ .

## Independences

- ▶  $[y \# w]$  denotes that the **current** value of  $y$  **must** be independent of the **initial** value of  $w$
- ▶  $T^\#$  ranges over set of **independences**

$T^\# \models T$  holds iff for all  $[y \# w] \in T^\#, t_1, t_2 \in T$ :

- ▶ if  $t_1$  and  $t_2$  agree on the **initial** value of all variables **except**  $w$
- ▶ then  $t_1$  and  $t_2$  **agree** on the **final** value of  $y$ .

Example: let  $T$  contain the two traces

$[w \mapsto (0, 0), y \mapsto (3, 2), z \mapsto (4, 5)],$   
 $[w \mapsto (7, 6), y \mapsto (3, 2), z \mapsto (4, 8)]$

Then  $[y \# w] \models T$  **does** hold;  $[z \# w] \models T$  **does not** hold

## A Hoare-like Logic

Judgements have the form  $G \vdash \{T^\#\} C \{T_0^\#\}$

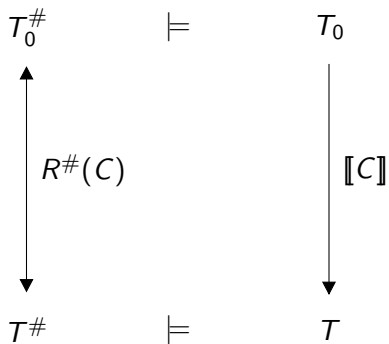
$G$  overapproximates **indirect** information flow. Consider

- ▶  $w := x + y ; \text{if } w < z \text{ then } C_1 \text{ else } C_2$
- ▶ in branches  $C_1$  and  $C_2$ ,  $G$  contains  $x, y, z$

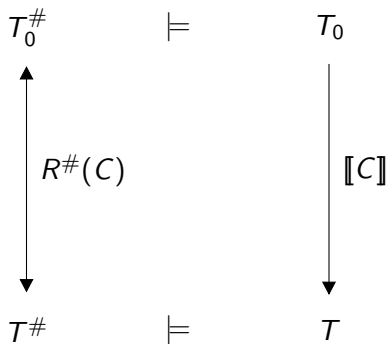
Sequential composition

$$\frac{G \vdash \{T_0^\#\} C_1 \{T_1^\#\} \quad G \vdash \{T_1^\#\} C_2 \{T_2^\#\}}{G \vdash \{T_0^\#\} C_1 ; C_2 \{T_2^\#\}}$$

## Recap



## Recap



$R^\#(C)$  is  $\emptyset \vdash \{T_0^\#\} C \{T^\#\}$

## Proof Rules

Assignment

$$G \vdash \{T_0^\#\} x := E \{T^\#\}$$

- ▶ if  $[y \# w] \in T^\#$  with  $y \neq x$  then  $[y \# w] \in T_0^\#$
- ▶ if  $[x \# w] \in T^\#$  then  $w \notin G$  and  $\forall z \in \text{fv}(E) \bullet [z \# w] \in T_0^\#$

Conditional

$$\frac{G \vdash \{T_0^\#\} C_1 \{T^\#\} \quad G \vdash \{T_0^\#\} C_2 \{T^\#\}}{G \vdash \{T_0^\#\} \text{if } E \text{ then } C_1 \text{ else } C_2 \{T^\#\}}$$

if  $w \notin G$  implies  $\forall x \in \text{fv}(E) \bullet [x \# w] \in T_0^\#$

## Proof Rules

Assignment

$$G \vdash \{T_0^\#\} x := E \{T^\#\}$$

- ▶ if  $[y \# w] \in T^\#$  with  $y \neq x$  then  $[y \# w] \in T_0^\#$
- ▶ if  $[x \# w] \in T^\#$  then  $w \notin G$  and  $\forall z \in \text{fv}(E) \bullet [z \# w] \in T_0^\#$

Conditional

$$\frac{G \vdash \{T_0^\#\} C_1 \{T^\#\} \quad G \vdash \{T_0^\#\} C_2 \{T^\#\}}{G \vdash \{T_0^\#\} \text{if } E \text{ then } C_1 \text{ else } C_2 \{T^\#\}}$$

if  $w \notin G$  implies  $\forall x \in \text{fv}(E) \bullet [x \# w] \in T_0^\#$

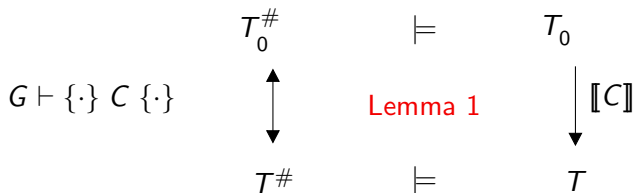
**NB:** given rule for **subtyping**, many equivalent formulations are possible (see paper)

## Example Derivation

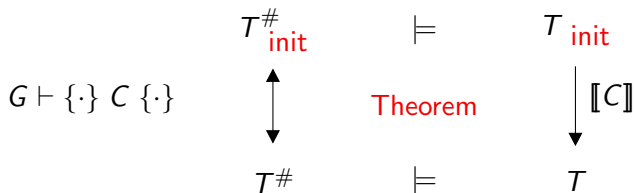
$[x \# y, z]$  abbreviates  $[x \# y], [x \# z]$ .

	$\{[l \# h, x], [h \# l, x], [x \# l, h]\}$
$x := h$	$\{[l \# h, x], [h \# l, x], [x \# l, x]\}$
if $x > 0$	( $G$ is now $\{h\}$ )
then $l := 7$	$\{[l \# x, l], [h \# l, x], [x \# l, x]\}$
else $x := 0$	$\{[l \# h, x], [h \# l, x], [x \# l, x]\}$
end of if	$\{[l \# x], [h \# l, x], [x \# l, x]\}$

## Correctness Results



## Correctness Results



Bootstrapping condition:

- ▶ for all  $t \in T_{\text{init}}$ , for all variables  $x$ :  
the **current** value of  $x$  in  $t$  **equals** its **initial** value
- ▶ if  $[x \# w] \in T^{\#}_{\text{init}}$  then  $x \neq w$

## Covert Channels

- ▶ We are **resource-insensitive**.

$$\emptyset \vdash \{ \{ [l \# h] \} \}$$

if  $h \neq 0$  then  $h := \text{CHEAP}$  else  $h := \text{EXPENSIVE}$

$$\{ \{ [l \# h] \} \}$$

An observer able to observe use of resources can detect whether  $h$  is 0 or not.

- ▶ We are even **termination-insensitive**:

$$\emptyset \vdash \{ \{ [l \# h] \} \} \text{ while } h \neq 0 \text{ do } h := 7 \{ \{ [l \# h] \} \}$$

Yet an observer able to observe non-termination can detect whether  $h$  is 0 or not.

## Existence of Strongest Postcondition

Any command  $C$  has a **strongest postcondition**

- ▶ given context  $G$  and precondition  $T_0^\#$
- ▶ there exists a set  $T^\#$  which is the **largest** with the property

$$G \vdash \{T_0^\#\} C \{T^\#\}$$

- ▶ the function  $sp(G, C, T_0^\#)$  computes that set.

Similarly, we can compute **weakest precondition**

## Computing Strongest Postcondition

$$sp(G, x := E, T^\#) = \\ \{[y \# w] \mid y \neq x \wedge [y \# w] \in T^\#\} \\ \cup \{[x \# w] \mid w \notin G \wedge \forall y \in \text{fv}(E) \bullet [y \# w] \in T^\#\}$$

$$sp(G, \text{if } E \text{ then } C_1 \text{ else } C_2, T^\#) = \\ \text{let } G_0 = G \cup \{w \mid \exists x \in \text{fv}(E) \bullet [x \# w] \notin T^\#\} \\ T_1^\# = sp(G_0, C_1, T^\#) \\ T_2^\# = sp(G_0, C_2, T^\#) \\ \text{in } T_1^\# \cap T_2^\#$$

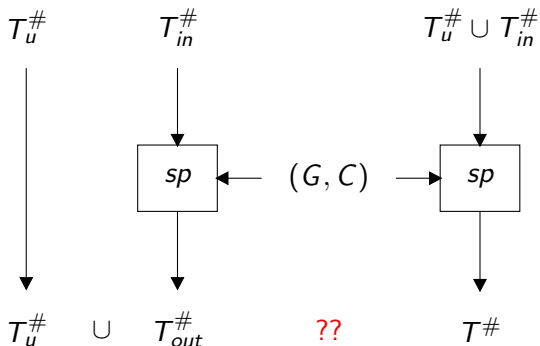
$$sp(G, \text{while } E \text{ do } C, T^\#) = \\ \dots \text{fixed point computation} \dots$$

## Example of Strongest Postcondition

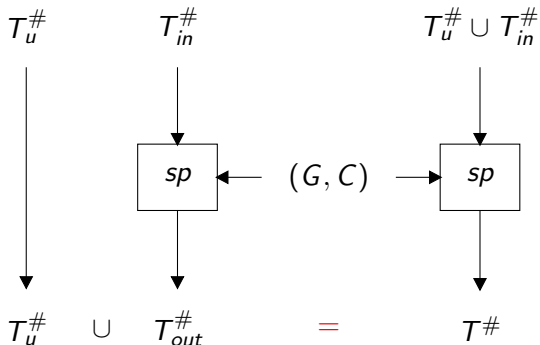
What you saw earlier, was computed using  $sp$

	$\{[l \# h, x], [h \# l, x], [x \# l, h]\}$
$x := h$	$\{[l \# h, x], [h \# l, x], [x \# l, x]\}$
if $x > 0$	( $G$ is now $\{h\}$ )
then $l := 7$	$\{[l \# x, l], [h \# l, x], [x \# l, x]\}$
else $x := 0$	$\{[l \# h, x], [h \# l, x], [x \# l, x]\}$
end of if	$\{[l \# x], [h \# l, x], [x \# l, x]\}$

## Modularity and Frame Rule

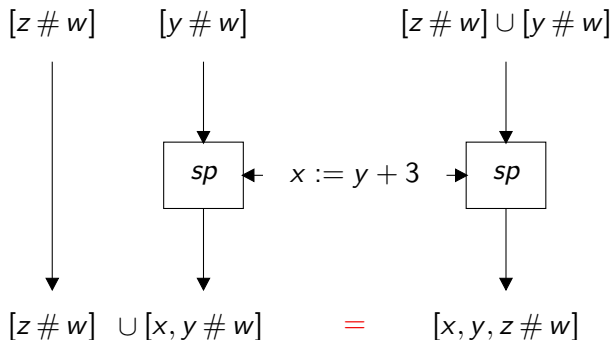


## Modularity and Frame Rule



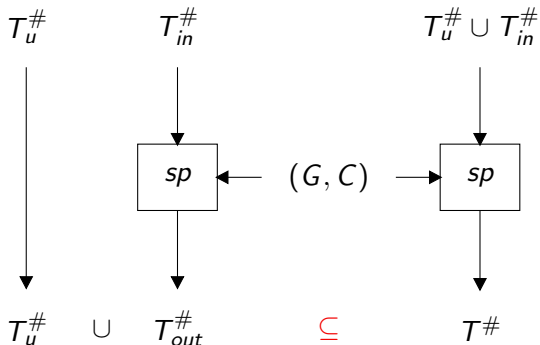
if  $lhs(T_u^\#) \cap fv(C) = \emptyset$

## Modularity and Frame Rule



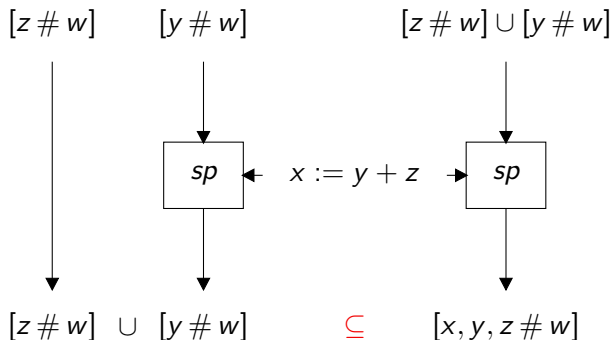
if  $lhs(T_u^\#) \cap fv(C) = \emptyset$

## Modularity and Frame Rule



if  $lhs(T_u^\#) \cap modified(C) = \emptyset$

## Modularity and Frame Rule



if  $lhs(T_u^\#) \cap modified(C) = \emptyset$

## Smith & Volpano = $[l \# h]$ is an invariant

### Their Type System

$$\frac{\Gamma, x : (T, \kappa) \vdash E : (T, \kappa)}{\Gamma, x : (T, \kappa) \vdash x := E : (\mathbf{com} \ \kappa)}$$

$$\frac{\Gamma \vdash E : (\mathbf{int}, \kappa) \quad \Gamma \vdash C_1 : (\mathbf{com} \ \kappa) \quad \Gamma \vdash C_2 : (\mathbf{com} \ \kappa)}{\Gamma \vdash \text{if } E \text{ then } C_1 \text{ else } C_2 : (\mathbf{com} \ \kappa)}$$

**Connection** A well-typed program has  $[l \# h]$  as **invariant**: if  $[l \# h]$  appears in the precondition, then it also appears in the strongest postcondition.

- ▶ we can handle  $l := h ; l := 0$

## Counterexamples

- ▶ if  $[l \# h]$  not in postcondition, we would like to find two different values of  $h$  producing different values of  $l$
- ▶ as analysis not complete, naive approach will produce **false positives**
- ▶ paper adopts semantic interpretation that makes all **false positives** become **genuine positives**

## Composition with other Analyses

**Parity** Assume an attacker can read the parity only. Then

```
while h do l := l + 2 ; h := h - 1
```

is secure. We can abstract it to

```
while h do h := h - 1
```

which our logic deems secure.

**Signs** Then

```
l := h × h + 1
```

is abstracted to

```
l := pos
```

which our logic deems secure.

## Related Work

Clark, Hankin & Hunt [Computer Languages, 2002]

- ▶ for each  $S$  determines whether different values for  $y$  prior to  $S$  results in different values for  $x$  after  $S$
- ▶ termination-sensitive

Joshi & Leino [SCP, 2000]

- ▶ semantic characterization of noninterference

$$C ; HH = HH ; C ; HH$$

where  $HH$  assigns an arbitrary value to a high variable.

- ▶ handles termination sensitivity/insensitivity

Darvas, Hähnle & Sands [WITS 2003]

- ▶ employs general purpose theorem prover
- ▶ can give counterexamples with actual runtime values

## Future Work

We want to investigate if our techniques are scalable  
(this is why we insisted on frame rule)

### Current directions

- ▶ extension with list pointers
- ▶ compile into machine code (translate assertions)

### Other Extensions

- ▶ functions, procedures, objects
- ▶ concurrency
- ▶ declassification

**Joint** with Sruthi Bandhakavi, Tamara Rezk, Franklyn Turbak; the  
Bandera group at Kansas State University