

# From Qualitative to Quantitative Program Analysis: Permissive Enforcement of Secure Information Flow

Mounir Assaf

Software Security Lab, CEA LIST, Saclay  
CIDre, Inria/Irisa/CentraleSupélec, Rennes  
Since 2015-09, Stevens Institute of Technology, Hoboken, US

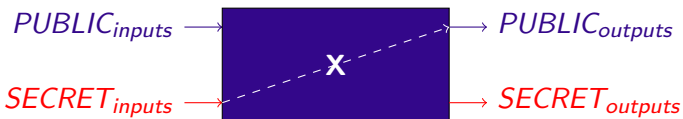


- **Information security :**
  - Confidentiality
  - Integrity
  - Availability
- **Traditionally, dissemination of information is prevented through Access control :**
  - Deals with what piece of information can be accessed? by whom?
  - Yet, is this piece of information handled correctly when accessed?
- **Information Flow Control :**
  - Tracks how information is propagated through a program
  - Verifies that **information flows** are secure with respect to a security policy

- Attacker model :
  - They know the source of programs and public outputs
  - They control public inputs



- Attacker model :
  - They know the source of programs and public outputs
  - They control public inputs



- A program is secure if non-interfering
  - Roughly, **non-interference** (in the case of confidentiality) is **independance of public outputs from secret inputs**

- **Termination-Insensitive Non-Interference (TINI)**
  - Two **terminating executions** which differ only on secret inputs deliver the same public outputs



- **Termination-Insensitive Non-Interference (TINI)**
  - Two **terminating executions** which differ only on secret inputs deliver the same public outputs



## Explicit flows

- produced when information is transferred directly from source to destination

**destination := source**

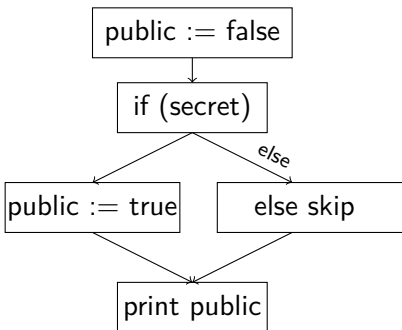
← - - - - -

Assignments generate explicit flows

- **Explicit flow** from variable source to destination

## Implicit flows

- produced when an assignment is **conditioned** on the value of an expression

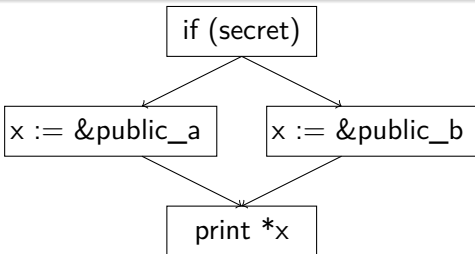


- Implicit flow** from variable `secret` to variable `public`



## Pointer-induced flows

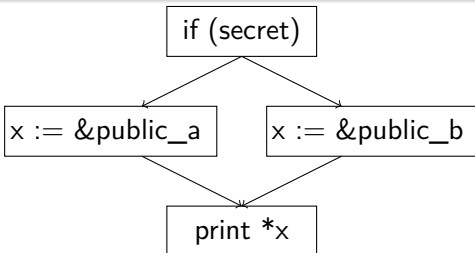
- produced whenever a pointer is dereferenced



- Attackers knowing the values of variables `public_a` and `public_b` can deduce information about variable `secret` when observing the output `*x`.

## Pointer-induced flows

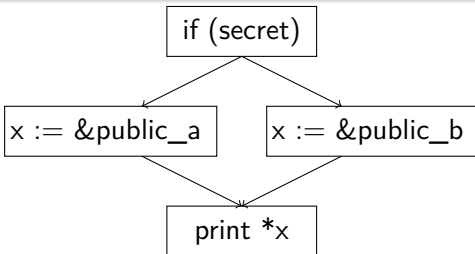
- produced whenever a pointer is dereferenced



- Attackers knowing the values of variables `public_a` and `public_b` can deduce information about variable `secret` when observing the output `*x`.
  - **Implicit flow** from `secret` to pointer `x`

## Pointer-induced flows

- produced whenever a pointer is dereferenced



- Attackers knowing the values of variables `public_a` and `public_b` can deduce information about variable `secret` when observing the output `*x`.
  - **Implicit flow** from `secret` to pointer `x`
  - **Pointer-induced flow** from pointer `x` to `*x`

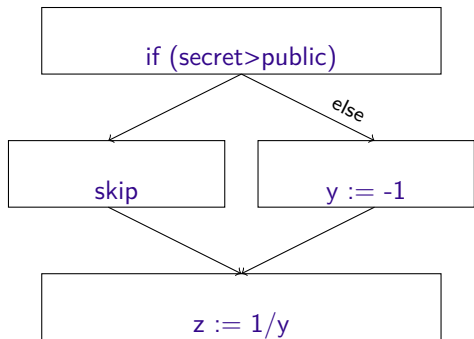
## I Information Flow

## II Qualitative IF

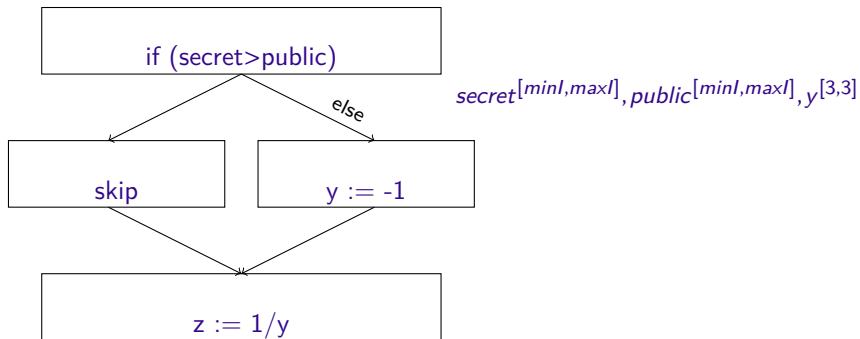
- 1 Value Analysis
- 2 Information Flow Control for C programs
- 3 PWhile monitor

## III Quantitative IF

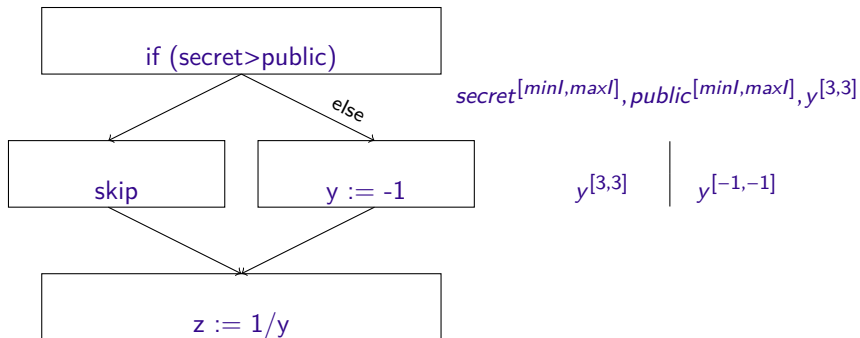
## IV Conclusion



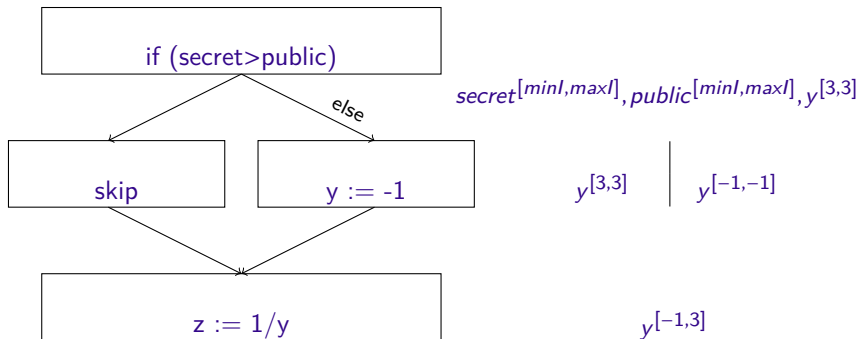
- **Frama-C**, an industrial-quality framework for source code analysis of C [Kirchner et al.,2015]
- **Value Analysis plug-in**
  - a tool based on Abstract Interpretation [Cousot & Cousot,77]
  - over-approximating the variation domains of variables
  - aimed at the detection of runtime errors



- **Frama-C**, an industrial-quality framework for source code analysis of C [Kirchner et al., 2015]
- **Value Analysis plug-in**
  - a tool based on Abstract Interpretation [Cousot & Cousot, 77]
  - over-approximating the variation domains of variables
  - aimed at the detection of runtime errors

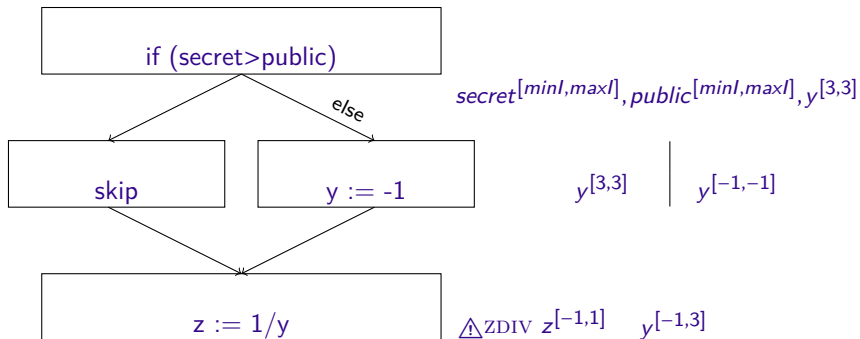


- **Frama-C**, an industrial-quality framework for source code analysis of C [Kirchner et al.,2015]
- **Value Analysis plug-in**
  - a tool based on Abstract Interpretation [Cousot & Cousot,77]
  - over-approximating the variation domains of variables
  - aimed at the detection of runtime errors

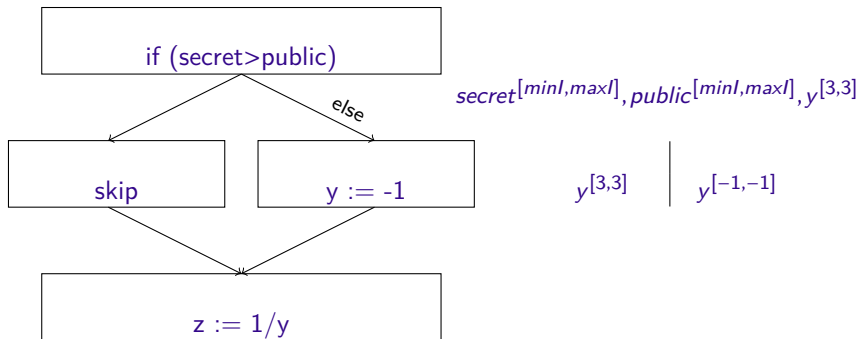


- **Frama-C**, an industrial-quality framework for source code analysis of C [Kirchner et al.,2015]
- **Value Analysis plug-in**
  - a tool based on Abstract Interpretation [Cousot & Cousot,77]
  - over-approximating the variation domains of variables
  - aimed at the detection of runtime errors

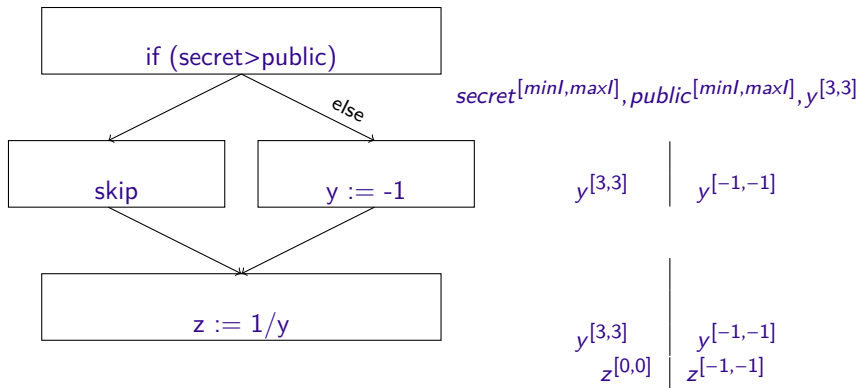




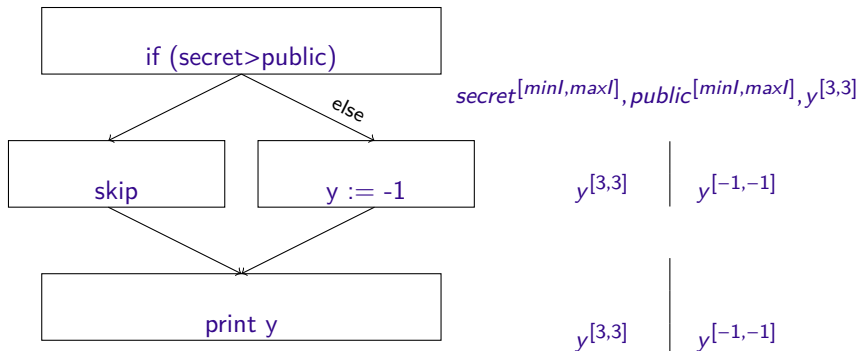
- **Frama-C**, an industrial-quality framework for source code analysis of C [Kirchner et al., 2015]
- **Value Analysis plug-in**
  - a tool based on Abstract Interpretation [Cousot & Cousot, 77]
  - over-approximating the variation domains of variables
  - aimed at the detection of runtime errors



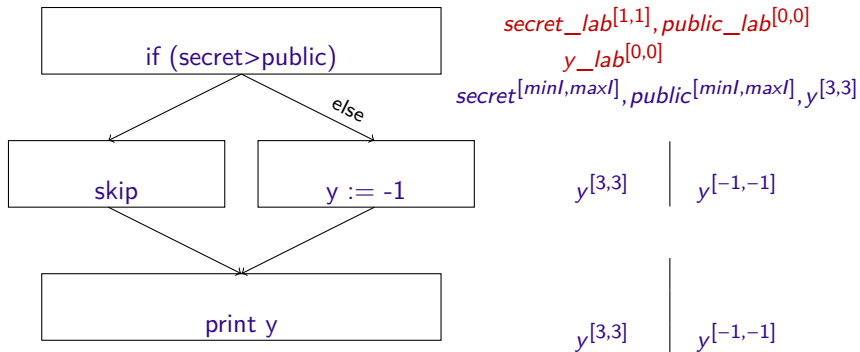
- Trace partitioning [Rival & Mauborgne, 05 & 07]
  - In a nutshell, simulating a partition of executions
  - Gaining precision by keeping both states separate



- Trace partitioning [Rival & Mauborgne, 05 & 07]
  - In a nutshell, simulating a partition of executions
  - Gaining precision by keeping both states separate

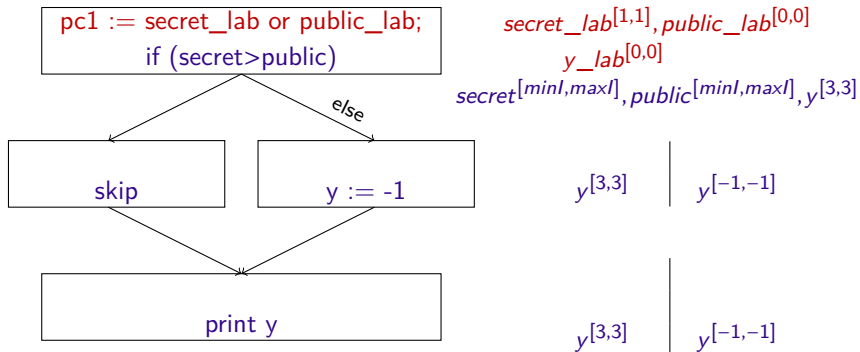


- The seed of an idea :
  - Instrument the source code of target programs according to traditional security type systems [Hunt & Sands,06]
  - Rely on Value Analysis for the computations



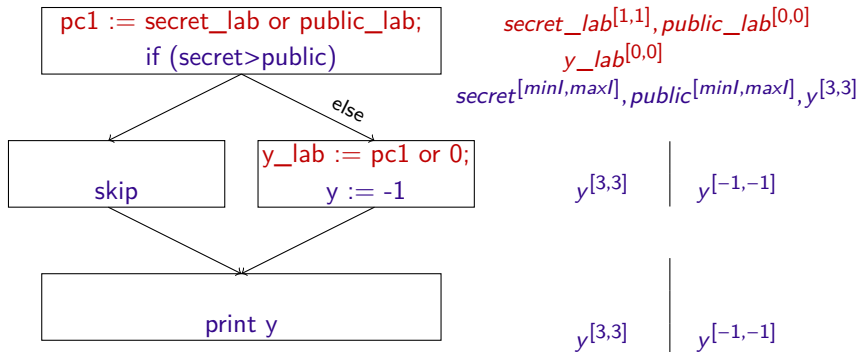
- The seed of an idea :

- Instrument the source code of target programs according to traditional security type systems [Hunt & Sands,06]
- Rely on Value Analysis for the computations
- Representing security labels:  $SECRET \triangleq 1$ ,  $PUBLIC \triangleq 0$   
union over security labels: logical or



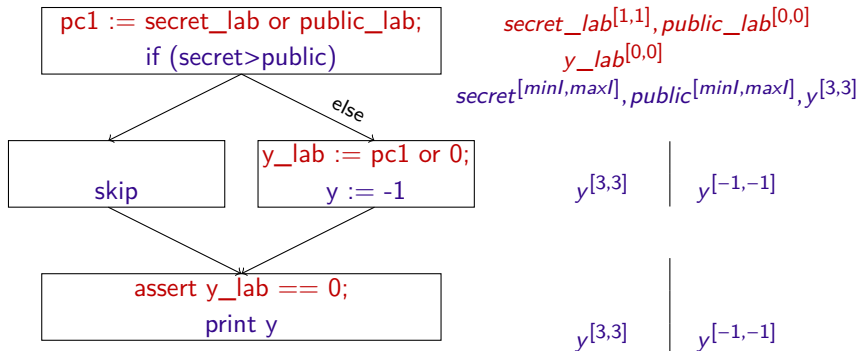
- The seed of an idea :

- Instrument the source code of target programs according to traditional security type systems [Hunt & Sands,06]
- Rely on Value Analysis for the computations
- Representing security labels:  $SECRET \triangleq 1$ ,  $PUBLIC \triangleq 0$   
union over security labels: logical or



- The seed of an idea :

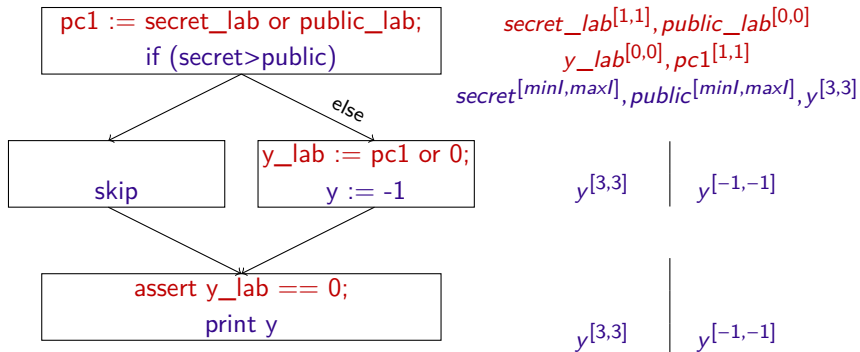
- Instrument the source code of target programs according to traditional security type systems [Hunt & Sands,06]
- Rely on Value Analysis for the computations
- Representing security labels:  $SECRET \triangleq 1$ ,  $PUBLIC \triangleq 0$   
union over security labels: logical or



- The seed of an idea :

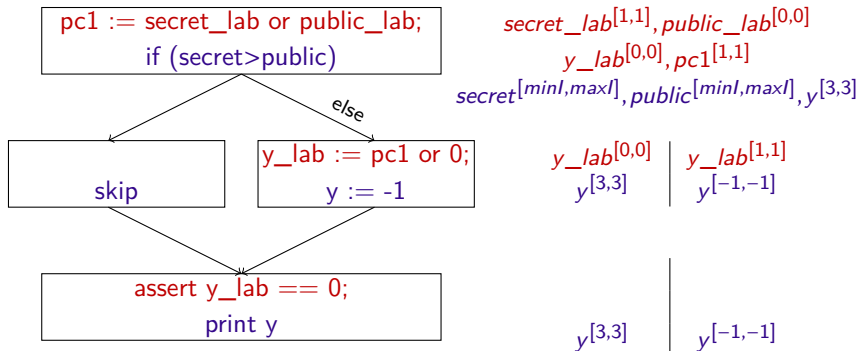
- Instrument the source code of target programs according to traditional security type systems [Hunt & Sands,06]
- Rely on Value Analysis for the computations
- Representing security labels:  $SECRET \triangleq 1, PUBLIC \triangleq 0$   
union over security labels: logical or





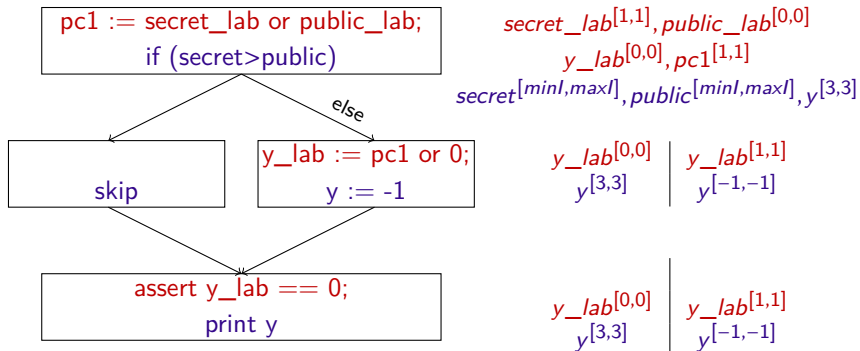
- The seed of an idea :

- Instrument the source code of target programs according to traditional security type systems [Hunt & Sands,06]
- Rely on Value Analysis for the computations
- Representing security labels:  $SECRET \triangleq 1$ ,  $PUBLIC \triangleq 0$   
union over security labels: logical or



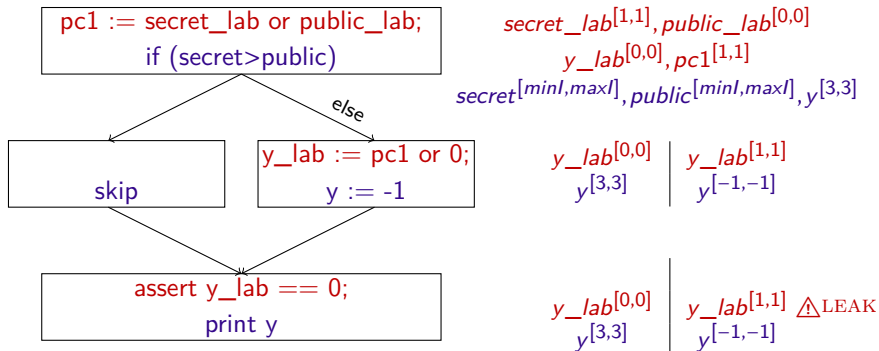
- The seed of an idea :

- Instrument the source code of target programs according to traditional security type systems [Hunt & Sands,06]
- Rely on Value Analysis for the computations
- Representing security labels:  $SECRET \triangleq 1$ ,  $PUBLIC \triangleq 0$   
union over security labels: logical or



- The seed of an idea :

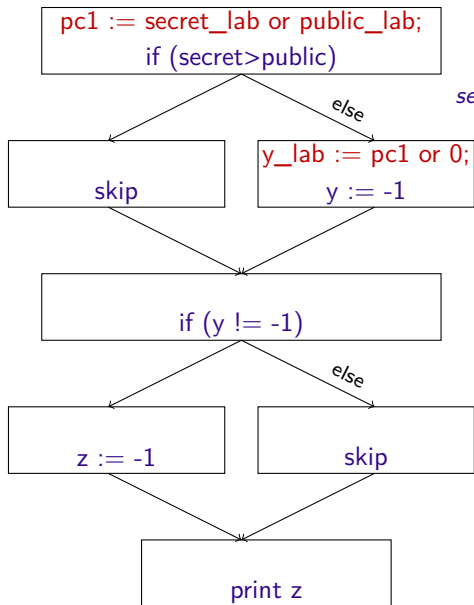
- Instrument the source code of target programs according to traditional security type systems [Hunt & Sands,06]
- Rely on Value Analysis for the computations
- Representing security labels:  $SECRET \triangleq 1$ ,  $PUBLIC \triangleq 0$   
union over security labels: logical or



- The seed of an idea :

- Instrument the source code of target programs according to traditional security type systems [Hunt & Sands,06]
- Rely on Value Analysis for the computations
- Representing security labels:  $SECRET \triangleq 1$ ,  $PUBLIC \triangleq 0$   
union over security labels: logical or

## Information Flow Control for C?

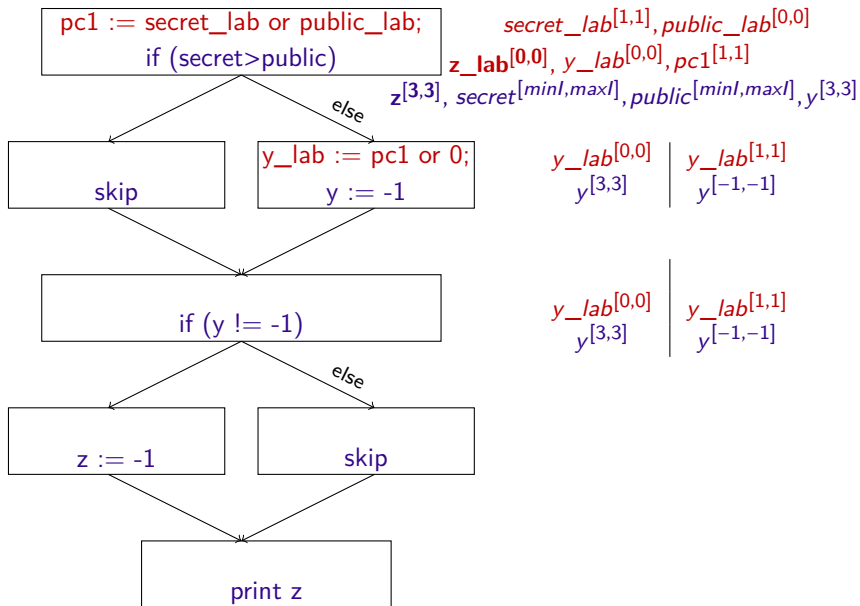


$secret\_lab^{[1,1]}, public\_lab^{[0,0]}$   
 $y\_lab^{[0,0]}, pc1^{[1,1]}$   
 $secret^{[minl,maxl]}, public^{[minl,maxl]}, y^{[3,3]}$

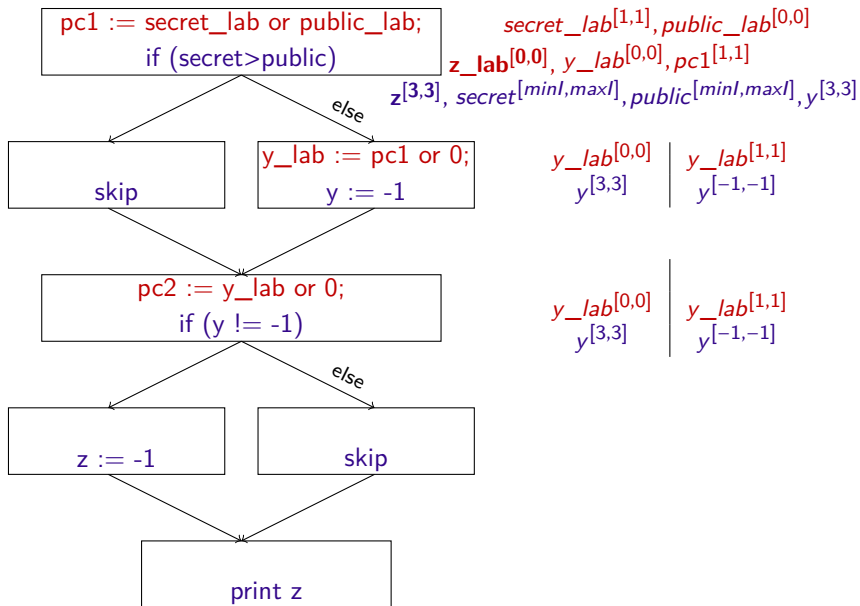
$y\_lab^{[0,0]}$  |  $y\_lab^{[1,1]}$   
 $y^{[3,3]}$  |  $y^{[-1,-1]}$

$y\_lab^{[0,0]}$  |  $y\_lab^{[1,1]}$   
 $y^{[3,3]}$  |  $y^{[-1,-1]}$

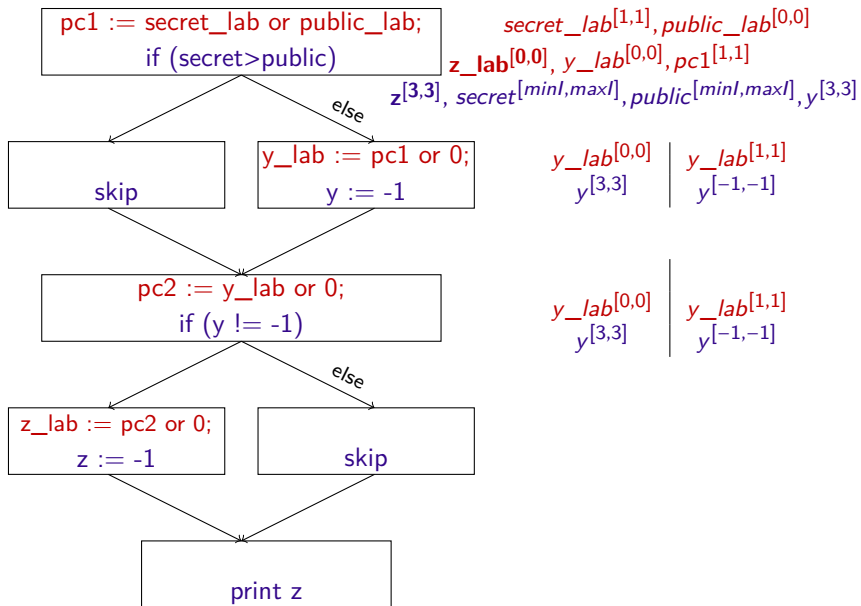
## Information Flow Control for C?



## Information Flow Control for C?

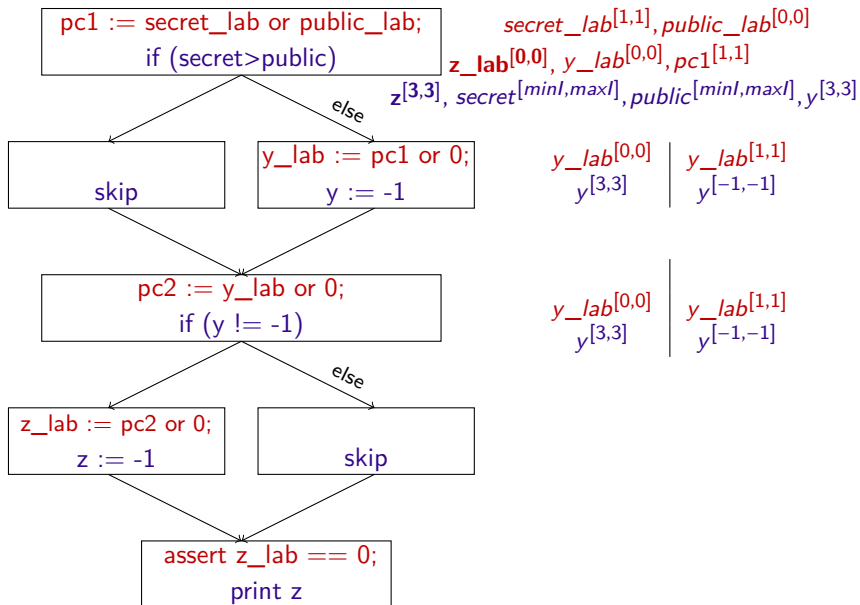


## Information Flow Control for C?

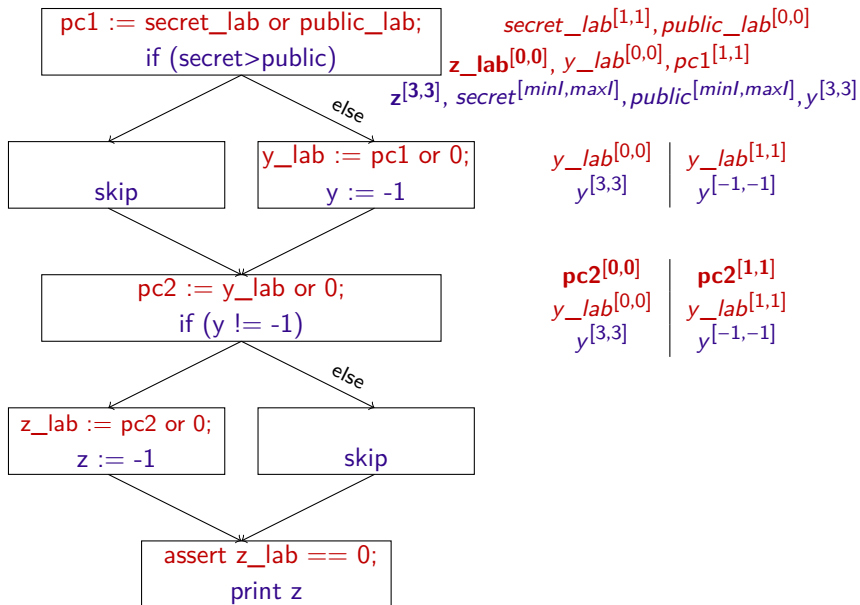




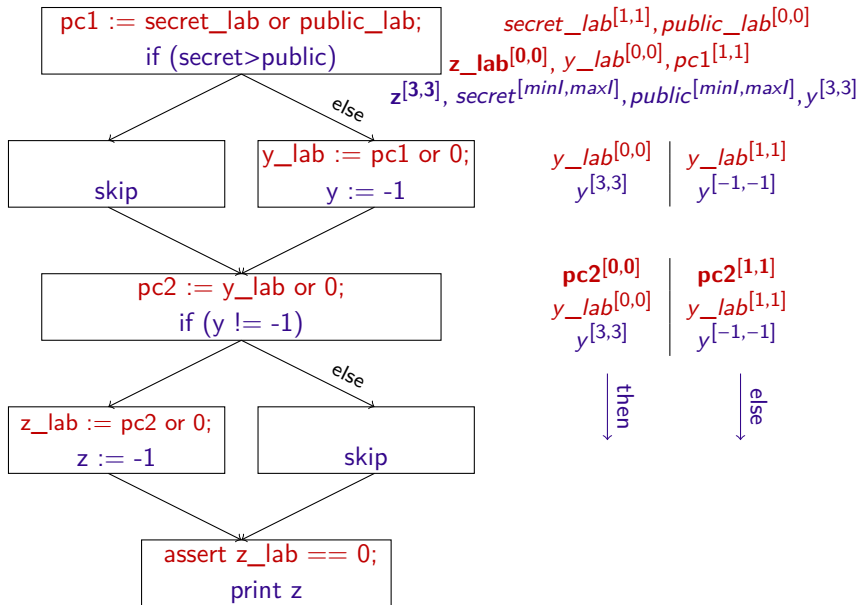
## Information Flow Control for C?



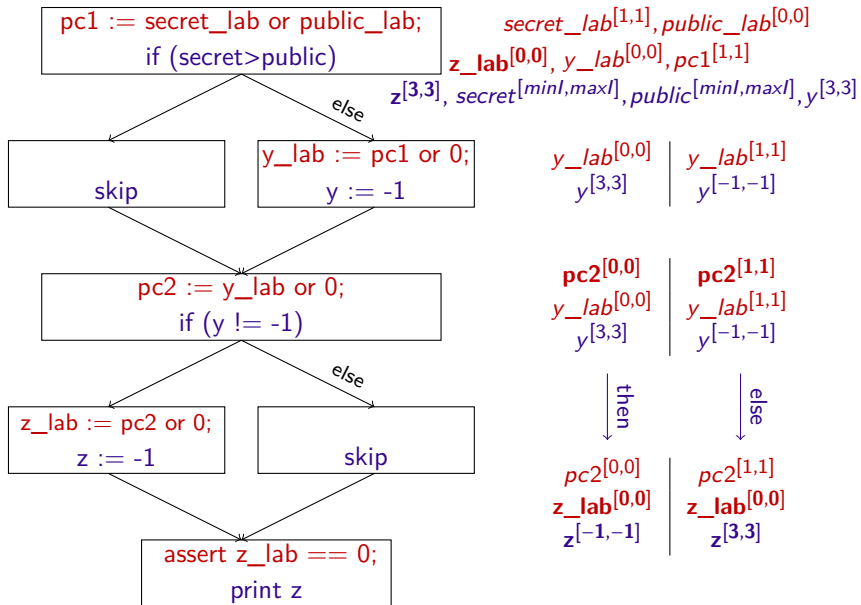
## Information Flow Control for C?



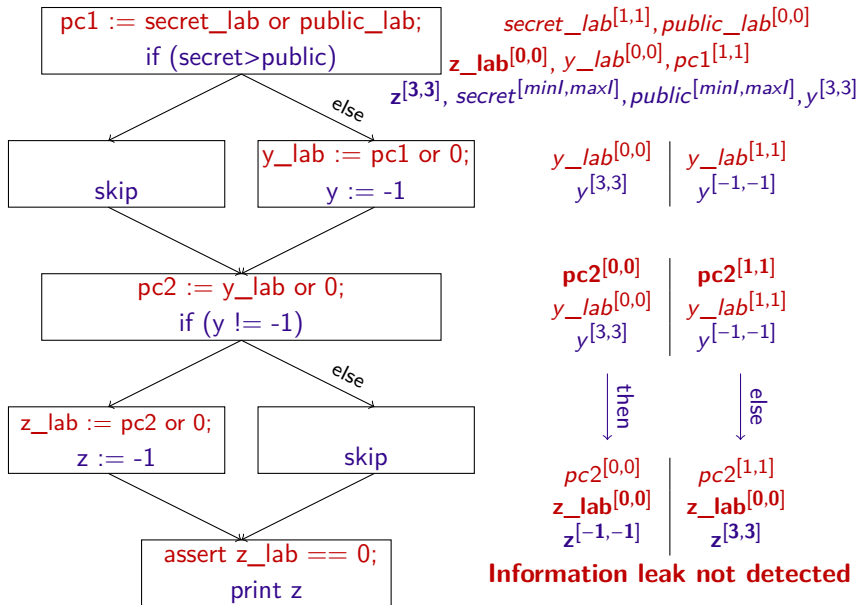
## Information Flow Control for C?



## Information Flow Control for C?



## Information Flow Control for C?



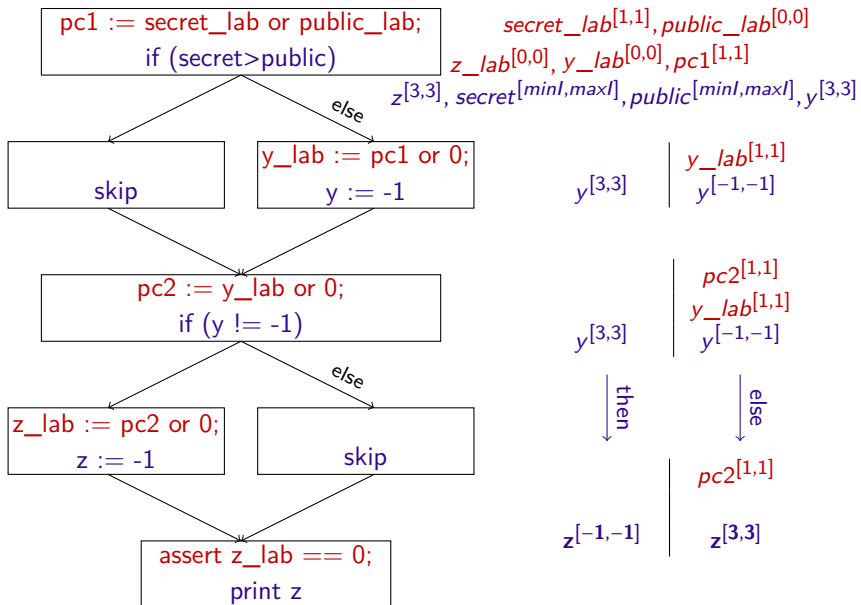
Non-interference is not a safety property [McLean,94] [Schneider,00]

Intuitively, even the fact that a variable is not modified during an execution may also leak sensitive information

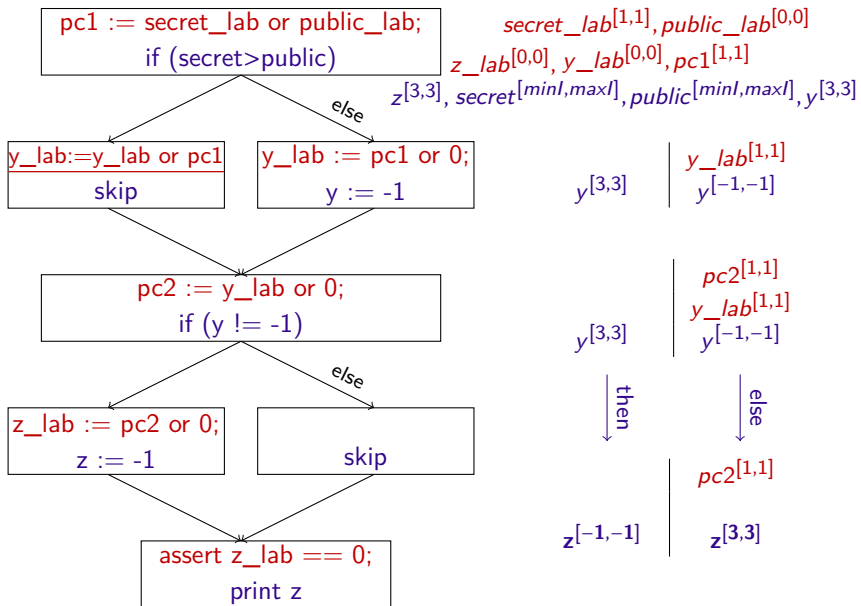
Information Flow Monitoring [Le Guernic et al.,06] [Russo & Sabelfeld,10]

A prior static analysis of target programs computing the set of modified variables in non-executed conditional branches

## IF monitoring to the Rescue!

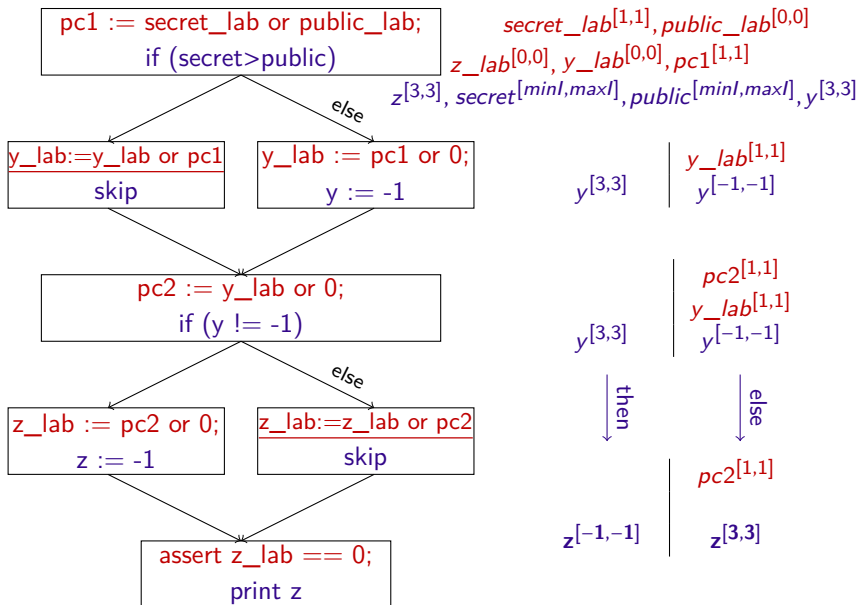


## IF monitoring to the Rescue!

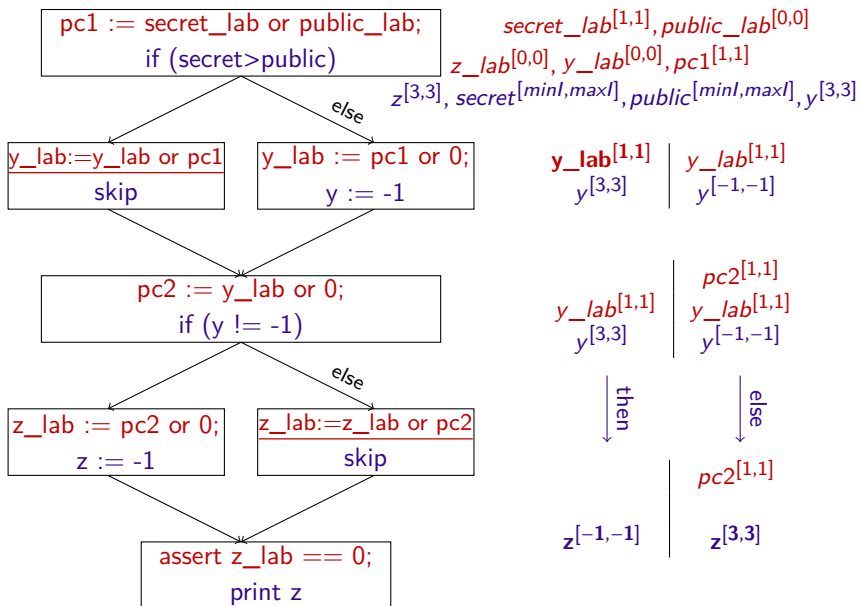




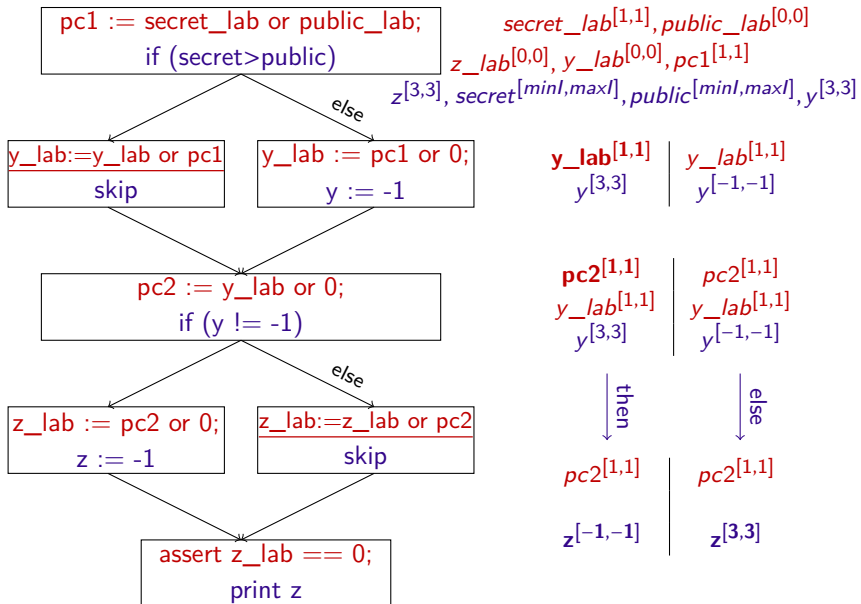
## IF monitoring to the Rescue!



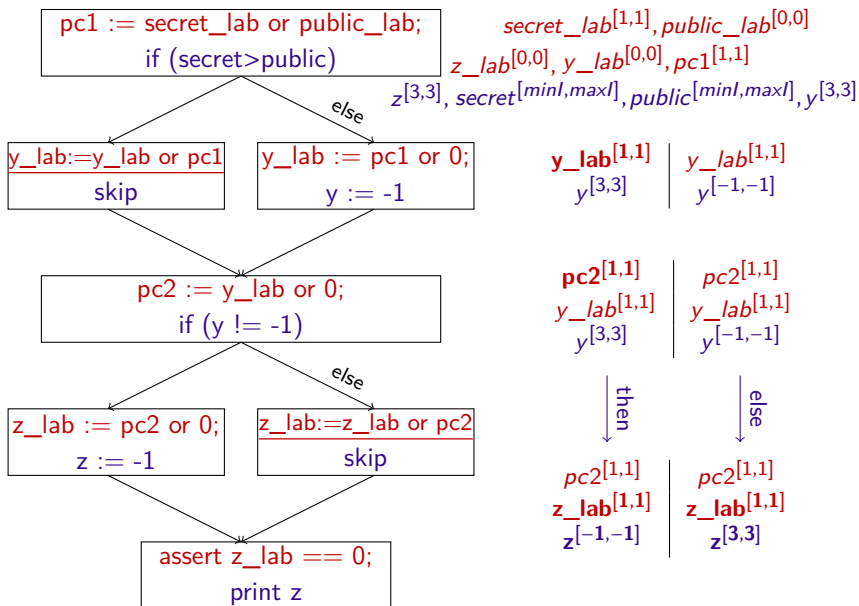
## IF monitoring to the Rescue!



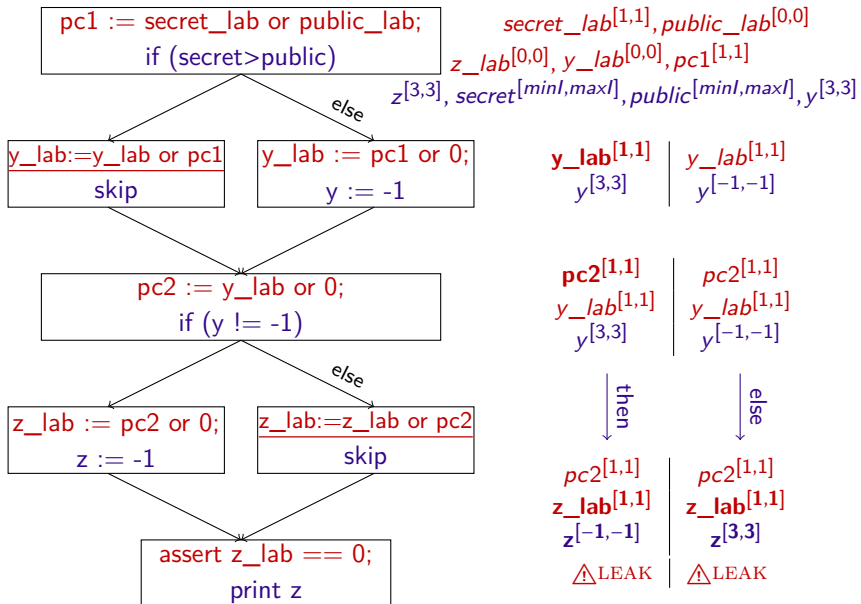
## IF monitoring to the Rescue!



## IF monitoring to the Rescue!



## IF monitoring to the Rescue!



```
1  int auth
2
3  int *leak
4
5
6  leak := &auth
7
8
9
10 print *leak
```

- **Aliasing invariant:** two expressions are aliased iff their auxiliary variables are aliased

```
1  int auth
2  label auth_lab
3  int *leak
4
5
6  leak := &auth
7
8
9
10 print *leak
```

- **Aliasing invariant:** two expressions are aliased iff their auxiliary variables are aliased

```
1  int auth
2  label auth_lab
3  int *leak
4  label leak_lab
5  label *leak_lab_d1
6  leak := &auth
7
8
9
10 print *leak
```

- **Aliasing invariant:** two expressions are aliased iff their auxiliary variables are aliased



```
1  int auth
2  label auth_lab
3  int *leak
4  label leak_lab
5  label *leak_lab_d1
6  leak := &auth
7  leak_lab := 0
8  leak_lab_d1 := &auth_lab
9
10 print *leak
```

- **Aliasing invariant:** two expressions are aliased iff their auxiliary variables are aliased

```
1  int auth
2  label auth_lab
3  int *leak
4  label leak_lab
5  label *leak_lab_d1
6  leak := &auth
7  leak_lab := 0
8  leak_lab_d1 := &auth_lab
9  assert leak_lab or *leak_lab_d1 == 0
10 print *leak
```

- **Aliasing invariant:** two expressions are aliased iff their auxiliary variables are aliased

- Our approach:
  - **Sound hybrid information flow monitor** for a language supporting pointers and aliasing
  - **Sound inlining approach** for our monitor  
[Assaf et al.,13a & 13b]
  - Implemented the inlining approach as a Frama-C plug-in: support of arrays, pointer arithmetics, functions, structures. . .
- Verifying TINI through:
  - Static analysis by relying on off-the-shelf tools, or
  - Monitoring
- **Future work:** a prior static analysis guiding the inlining approach to support casts, unions and dynamic allocation. . .

# I Information Flow

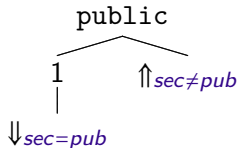
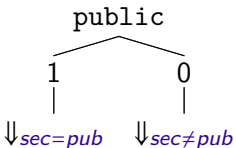
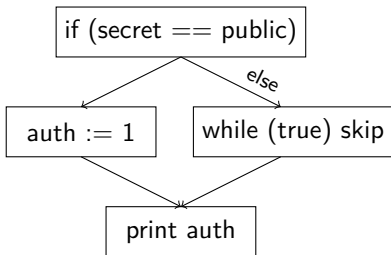
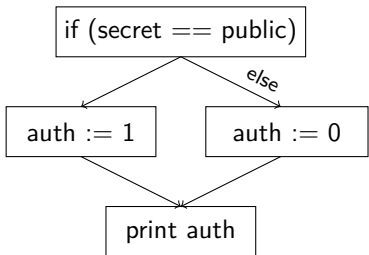
## II Qualitative IF

## III Quantitative IF

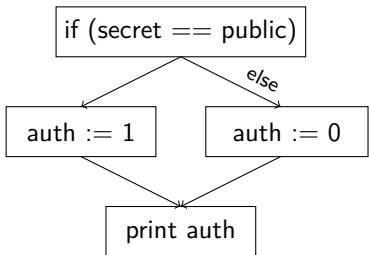
- 1 Relative Secrecy
- 2 Cardinal Abstraction
- 3 Tree Abstraction

## IV Conclusion

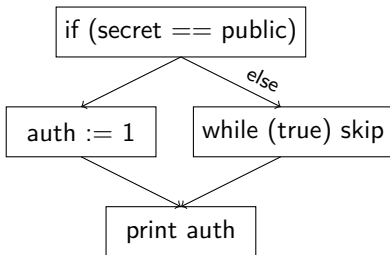
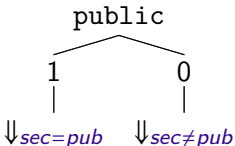
## Termination-Insensitive Non-interference



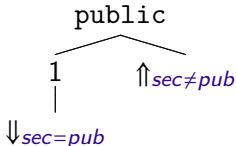
- From attackers' perspective: **both programs are equivalent**



⚠ Interferent program



Non-interferent program



- From attackers' perspective: **both programs are equivalent**

- Information leaks due to the observation of divergence are discarded
  - The probability of **polynomial time attackers** guessing the secret is **negligible** in the size of the secret [Askarov et al.,08]
  - **Let us prove security of programs wrt. this exact assumption**
  - A flavour of Relative Secrecy [Volpano and Smith,00]

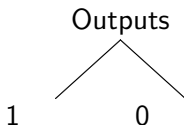
The framework of min-entropy/min-capacity [Smith, 09 & 11]

Quantifying the leakage wrt. the probability of attackers guessing the secret

```

1  if (secret == 0) {
2      print 1 }
3  else {
4      print 0 }

```



- What's the probability of attackers guessing the secret after observing an output? ( $secret \in [0, 2^N - 1]$ )



```

1  if (secret == 0) {
2      print 1 }
3  else {
4      print 0 }

```

Outputs

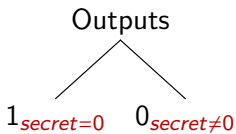
1<sub>secret=0</sub>    0<sub>secret≠0</sub>

- What's the probability of attackers guessing the secret after observing an output? ( $secret \in [0, 2^N - 1]$ )

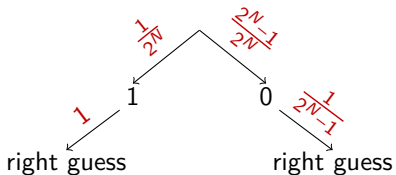
```

1  if (secret == 0) {
2      print 1 }
3  else {
4      print 0 }

```



- What's the probability of attackers guessing the secret after observing an output? ( $secret \in [0, 2^N - 1]$ )

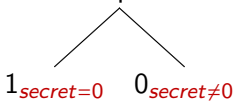


```

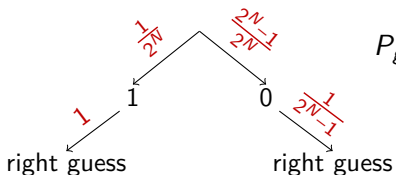
1  if (secret == 0) {
2      print 1 }
3  else {
4      print 0 }

```

Outputs

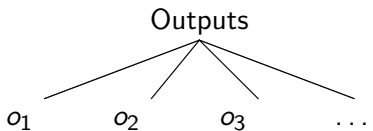


- What's the probability of attackers guessing the secret after observing an output? ( $secret \in [0, 2^N - 1]$ )

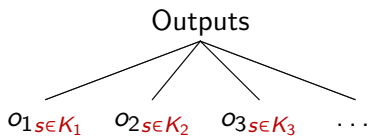


$$\begin{aligned}
 P_{\text{guess}} &= \frac{1}{2^N} \times \frac{1}{1} + \frac{2^N - 1}{2^N} \times \frac{1}{2^N - 1} \\
 &= \frac{2}{2^N}
 \end{aligned}$$

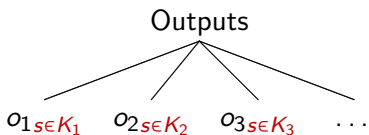
- A pattern :



- A pattern :

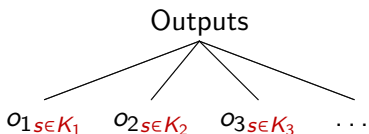


- A pattern :



$$\begin{aligned}
 P_{\text{guess}} &= \sum_{o_i \in \text{Outputs}} \frac{|K_i|}{2^N} \times \frac{1}{|K_i|} \\
 &= \frac{|\text{Outputs}|}{2^N}
 \end{aligned}$$

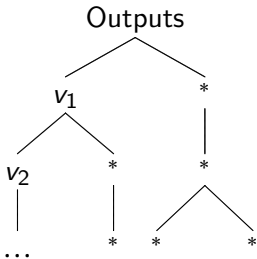
- A pattern :



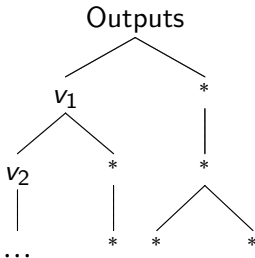
$$\begin{aligned}
 P_{\text{guess}} &= \sum_{o_i \in \text{Outputs}} \frac{|K_i|}{2^N} \times \frac{1}{|K_i|} \\
 &= \frac{|\text{Outputs}|}{2^N}
 \end{aligned}$$

- The leakage of a program in bits, assuming a uniform distribution  $\vec{\pi}_u$  of the secret: [Smith,09 & 11]

$$\begin{aligned}
 \mathcal{L}_{\vec{\pi}_u} &\triangleq \text{Initial uncertainty} - \text{Remaining uncertainty} \\
 &= -\log_2 \frac{1}{2^N} - \left( -\log_2 \frac{|\text{Outputs}|}{2^N} \right) \\
 &= \log_2 |\text{Outputs}|
 \end{aligned}$$

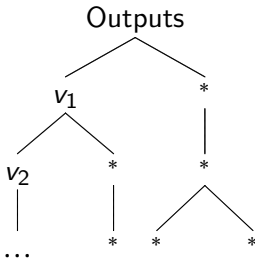






- **Polynomial time attackers** observe only a polynomial amount of outputs:

$$o_i = v_1 \cdot v_2 \cdot v_3 \dots \text{ such that } \text{length}(o_i) \leq b, \text{ with } \log(b) = o(N)$$



- **Polynomial time attackers** observe only a polynomial amount of outputs:

$$o_i = v_1 \cdot v_2 \cdot v_3 \dots \text{ such that } \text{length}(o_i) \leq b, \text{ with } \log(b) = o(N)$$

- **Relative Secrecy** for deterministic programs:

$$\log_2 |\text{Outputs}_b| = o(N) \text{ when } \log(b) = o(N)$$

- A problem : how do we count the cardinal of outputs?  
( $\mathcal{ML} = \log_2 |\text{Outputs}_b|$ )

```

1  x := s mod 4
2  print x

```

```

1  x := s mod 4
2  print x

```

- A problem : how do we count the cardinal of outputs?  
( $\mathcal{ML} = \log_2 |\text{Outputs}_b|$ )

1  
2

```
x := s mod 4
print x
```

1  $s \mapsto 0, s \mapsto 1, \dots, s \mapsto 2^N - 1$   
2  $x \mapsto 0, x \mapsto 1, x \mapsto 2, x \mapsto 3$

- finding the set of final states,

1  
2

```
x := s mod 4
print x
```

- A problem : how do we count the cardinal of outputs?  
( $\mathcal{ML} = \log_2 |\text{Outputs}_b|$ )

1  
2

```
x := s mod 4
print x
```

1  $s \mapsto 0, s \mapsto 1, \dots, s \mapsto 2^N - 1$   
2  $x \mapsto 0, x \mapsto 1, x \mapsto 2, x \mapsto 3$

- finding the set of final states,
- counting them !

1  
2

```
x := s mod 4
print x
```

- A problem : how do we count the cardinal of outputs?  
( $\mathcal{ML} = \log_2 |\text{Outputs}_b|$ )

1  
2

```
x := s mod 4
print x
```

1  $s \mapsto 0, s \mapsto 1, \dots, s \mapsto 2^N - 1$   
2  $x \mapsto 0, x \mapsto 1, x \mapsto 2, x \mapsto 3$

- finding the set of final states, **⚠ not computable**
- counting them !

1  
2

```
x := s mod 4
print x
```

- A problem : how do we count the cardinal of outputs?  
( $\mathcal{ML} = \log_2 |\text{Outputs}_b|$ )

1  
2

```
x := s mod 4
print x
```

1  $s \mapsto 0, s \mapsto 1, \dots, s \mapsto 2^N - 1$   
 2  $x \mapsto 0, x \mapsto 1, x \mapsto 2, x \mapsto 3$

- finding the set of final states, **⚠ not computable**
- counting them !
- A theory of sound semantics approximations : **abstract interpretation**

1  
2

```
x := s mod 4
print x
```

1  $s^{[0, 2^N - 1]}$   
 2  $x^{[0, 3]}$

$\mathcal{ML} = \log_2(4)$   
 $= 2 \text{ bits}$

- Over-approximating the set of reachable final states : **a fairly large body of the literature in abstract interpretation ...**
  - quantifying information flow by a generic abstract interpretation (over-approximated bounds) and symbolic execution (under-approximated bounds)  
[Köpf and Rybalchenko, 2010 & 2013]



- Over-approximating the set of reachable final states : **a fairly large body of the literature in abstract interpretation ...**
  - quantifying information flow by a generic abstract interpretation (over-approximated bounds) and symbolic execution (under-approximated bounds)  
[Köpf and Rybalchenko, 2010 & 2013]
- First limitation :

```
1 x := s mod 2
2 x := x + input
3 print x
```

- Over-approximating the set of reachable final states : **a fairly large body of the literature in abstract interpretation ...**
  - quantifying information flow by a generic abstract interpretation (over-approximated bounds) and symbolic execution (under-approximated bounds)  
[Köpf and Rybalchenko, 2010 & 2013]
- First limitation :

```

1 x := s mod 2
2 x := x + input
3 print x

```

$$0 \quad s \mapsto [0, 2^N - 1], \text{input} \mapsto [0, 2^N - 1]$$

$$1 \quad x \mapsto [0, 1]$$

$$2 \quad x \mapsto [0, 2^N - 1]$$

$$\mathcal{ML} = \log_2 2^N = N \text{ bits? no!}$$

- Over-approximating the set of reachable final states : **a fairly large body of the literature in abstract interpretation ...**
  - quantifying information flow by a generic abstract interpretation (over-approximated bounds) and symbolic execution (under-approximated bounds)  
[Köpf and Rybalchenko, 2010 & 2013]
- First limitation : **public inputs?**

```

1 x := s mod 2
2 x := x + input
3 print x

```

$$0 \quad s \mapsto [0, 2^N - 1], \text{input} \mapsto [0, 2^N - 1]$$

$$1 \quad x \mapsto [0, 1]$$

$$2 \quad x \mapsto [0, 2^N - 1]$$

$$\mathcal{ML} = \log_2 2^N = N \text{ bits? no!}$$

- Over-approximating the set of reachable final states : **a fairly large body of the literature in abstract interpretation ...**
  - quantifying information flow by a generic abstract interpretation (over-approximated bounds) and symbolic execution (under-approximated bounds)  
[Köpf and Rybalchenko, 2010 & 2013]
- First limitation : **public inputs?**

```

1 x := s mod 2
2 x := x + input
3 print x

```

$$0 \quad s \mapsto [0, 2^N - 1], \text{input} \mapsto [0, 2^N - 1]$$

$$1 \quad x \mapsto [0, 1]$$

$$2 \quad x \mapsto [0, 2^N - 1]$$

$$\mathcal{ML} = \log_2 2^N = N \text{ bits? no!}$$

- Second limitation : **intermediate outputs?**

- For each attacker-controlled low input, we have a channel – a sub-program – possibly leaking information

```

1 x := s mod 2
2 x := x + 1
3 print x

```

```

1 x := s mod 2
2 x := x + 3
3 print x

```

...

```

1 x := s mod 2
2 x := x + 2
3 print x

```

```

1 x := s mod 2
2 x := x + 4
3 print x

```

In fact, for each one of these sub-programs :

$$\mathcal{ML} = \log_2 |\text{Outputs}| = \log_2 2 = 1 \text{ bit}$$

- Compute the cardinal of reachable states – of outputs – for each possible public input?

	sub-program 0	sub-program 1	...	sub-program $2^N - 1$
$pp_1$	$s^{[0,2^N-1]} \underline{input}^{[0,0]}$	$s^{[0,2^N-1]} \underline{input}^{[1,1]}$		$s^{[0,2^N-1]} \underline{input}^{[2^N-1,2^N-1]}$
$pp_2$	$x^{[0,1]} s^{[0,2^N-1]} \underline{input}^{[0,0]}$	$x^{[0,1]} s^{[0,2^N-1]} \underline{input}^{[1,1]}$		$x^{[0,1]} s^{[0,2^N-1]} \underline{input}^{[2^N-1,2^N-1]}$
$pp_3$	$x^{[0,1]} s^{[0,2^N-1]} \underline{input}^{[0,0]}$	$x^{[1,2]} s^{[0,2^N-1]} \underline{input}^{[1,1]}$		$x^{[2^N-1,2^N]} s^{[0,2^N-1]} \underline{input}^{[2^N-1,2^N]}$
#	2	2		2

- Compute the cardinal of reachable states – of outputs – for each possible public input? **computationally inefficient!**

	sub-program 0	sub-program 1	...	sub-program $2^N - 1$
$pp_1$	$s^{[0,2^N-1]} \underline{input^{[0,0]}}$	$s^{[0,2^N-1]} \underline{input^{[1,1]}}$		$s^{[0,2^N-1]} \underline{input^{[2^N-1,2^N-1]}}$
$pp_2$	$x^{[0,1]} s^{[0,2^N-1]} \underline{input^{[0,0]}}$	$x^{[0,1]} s^{[0,2^N-1]} \underline{input^{[1,1]}}$		$x^{[0,1]} s^{[0,2^N-1]} \underline{input^{[2^N-1,2^N-1]}}$
$pp_3$	$x^{[0,1]} s^{[0,2^N-1]} \underline{input^{[0,0]}}$	$x^{[1,2]} s^{[0,2^N-1]} \underline{input^{[1,1]}}$		$x^{[2^N-1,2^N]} s^{[0,2^N-1]} \underline{input^{[2^N-1,2^N]}}$
#	2	2		2

- Let us abstract more. . . And keep only the cardinal of values

$$pp_1 : s^{2^N} \quad input^1$$

# I Information Flow

## II Qualitative IF

## III Quantitative IF

- 1 Relative Secrecy
- 2 Cardinal Abstraction
- 3 Tree Abstraction

## IV Conclusion



- A dedicated abstract domain computing an over-approximation of the cardinal of values a variable can take when attackers provide public inputs

```
1 x := s mod 2
2 x := x * input
3 print x
```

- A first step to quantify information flow for programs with low inputs
- Towards the combination of multiple domains to refine the **cardinal abstraction**

- A dedicated abstract domain computing an over-approximation of the cardinal of values a variable can take when attackers provide public inputs

```

1 x := s mod 2
2 x := x * input
3 print x

```

0  $s \mapsto (\{pp_0\}, 2^N), input \mapsto (\{pp_0\}, 1)$

1  $x \mapsto (\{pp_1\}, 2)$

2  $x \mapsto (\{pp_2\}, 2)$

$\mathcal{ML} = \log_2(2) = 1 \text{ bit}$

- A first step to quantify information flow for programs with low inputs
- Towards the combination of multiple domains to refine the **cardinal abstraction**

# I Information Flow

## II Qualitative IF

## III Quantitative IF

- ① Relative Secrecy
- ② Cardinal Abstraction
- ③ Tree Abstraction

## IV Conclusion

```

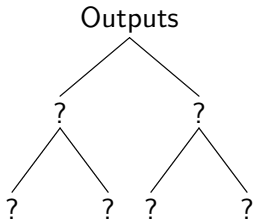
1 x := s mod 2
2 print x
3 x := s * input
4 print x

```

1  $x \mapsto (\{pp_1\}, 2)$

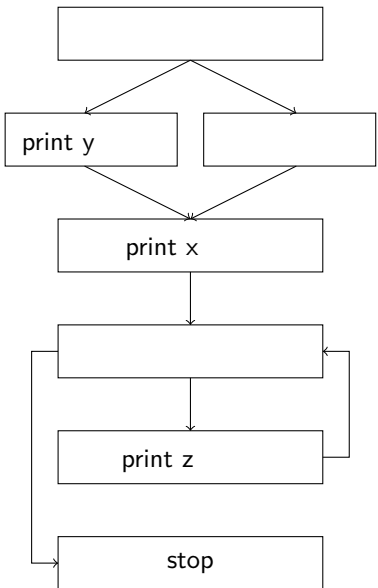
3  $x \mapsto (\{pp_3\}, 2)$

- Computing a **regular specification** representing attackers' observations

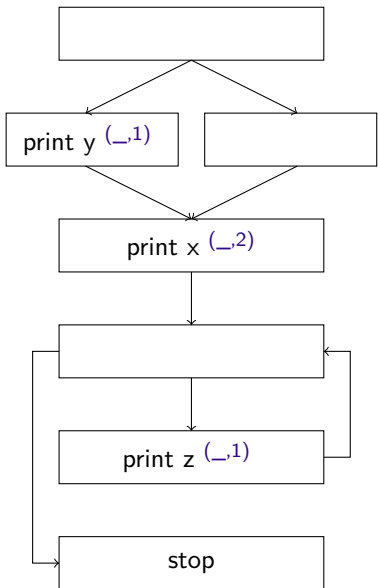


- $\mathcal{O}^2 \triangleq$  an atomic combinatorial class containing 2 undefined observations of size 1
- Attackers' observations :

$$\mathcal{C} = \mathcal{O}^2 \cdot \mathcal{O}^2$$

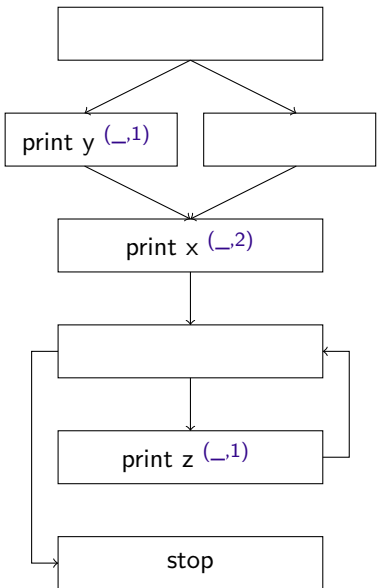


- Regular specifications:
  - concatenation  $\cdot$
  - combinatorial sum  $\oplus$
  - sequence construction  $\star$



- Regular specifications:

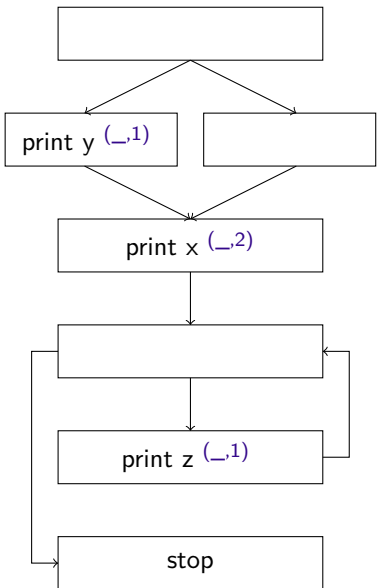
- concatenation  $\cdot$
- combinatorial sum  $\oplus$
- sequence construction  $\star$



- Regular specifications:
  - concatenation  $\cdot$
  - combinatorial sum  $\oplus$
  - sequence construction  $\star$

- Attackers' observations:

$$\mathcal{C} = (\mathcal{O}^1 \oplus \mathcal{O}^\epsilon)$$

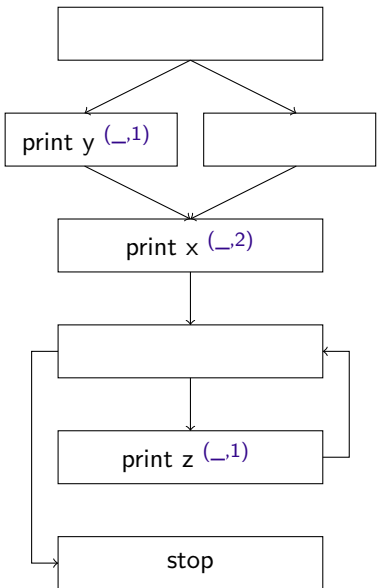


- Regular specifications:
  - concatenation  $\cdot$
  - combinatorial sum  $\oplus$
  - sequence construction  $\star$

- Attackers' observations:

$$\mathcal{C} = (\mathcal{O}^1 \oplus \mathcal{O}^\epsilon) \cdot \mathcal{O}^2$$

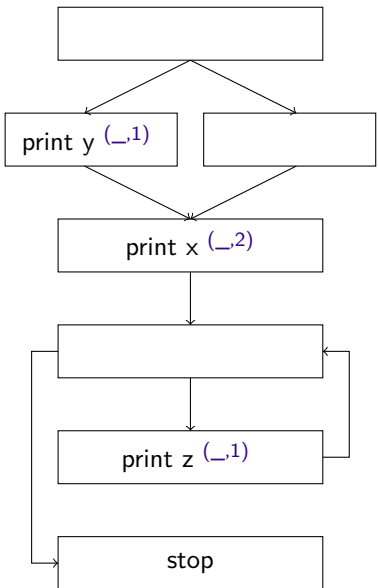




- Regular specifications:
  - concatenation  $\cdot$
  - combinatorial sum  $\oplus$
  - sequence construction  $\star$

- Attackers' observations:

$$\mathcal{C} = (\mathcal{O}^1 \oplus \mathcal{O}^\epsilon) \cdot \mathcal{O}^2 \cdot (\mathcal{O}^1)^\star$$



- Regular specifications:
  - concatenation  $\cdot$
  - combinatorial sum  $\oplus$
  - sequence construction  $\star$

- Attackers' observations:

$$\mathcal{C} = (\mathcal{O}^1 \oplus \mathcal{O}^\epsilon) \cdot \mathcal{O}^2 \cdot (\mathcal{O}^1)^\star \cdot \mathcal{O}^\uparrow$$

- Relying on **Analytic Combinatorics** to systematically translate **regular specifications** into **Ordinary Generating Functions (OGF)** [Flajolet and Sedgewick,09]

$$\mathcal{C} = (\mathcal{O}^1 \oplus \mathcal{O}^\epsilon) \cdot \mathcal{O}^2 \cdot (\mathcal{O}^1)^\star \cdot \mathcal{O}^\uparrow$$

$$C(z) = (z+1) \cdot 2z \cdot \left( \frac{1}{1-z} \right) \cdot z$$

- Extracting an **asymptotic** estimation of the leakage for **polynomial time attackers** by studying the singularities of the computed **OGF**
- Derived sufficient conditions on the **OGF** to prove **Relative Secrecy**

## Qualitative IF

- **PWhile Monitor** : A sound hybrid information flow monitor supporting pointers and aliasing
- A **sound inlining approach**

## Quantitative IF

- **Relative Secrecy** : Relaxing TINI while providing the same security guarantees wrt. polynomial time attackers
- **Cardinal Abstraction** : a sound analysis over-approximating the leakage for batch computations with public inputs
- **Tree Abstraction** : computing a regular specification of attackers' observations and deriving sufficient conditions to prove **Relative Secrecy**

## Qualitative IF

- **PWhile Monitor** : A sound hybrid information flow monitor supporting pointers and aliasing
- A **sound inlining approach** ✓

## Quantitative IF

- **Relative Secrecy** : Relaxing TINI while providing the same security guarantees wrt. polynomial time attackers
- **Cardinal Abstraction** : a sound analysis over-approximating the leakage for batch computations with public inputs ✓
- **Tree Abstraction** : computing a regular specification of attackers' observations and deriving sufficient conditions to prove **Relative Secrecy**

## PWhile Monitor

Overcoming an inherent limitation of the inlining approach: casts, dynamic allocation. . .

## Relative Secrecy

Improving both the cardinal abstraction and the tree abstraction by combining different abstract domains :

- Numerical abstractions
- Selective Trace Partitioning
- Relational abstraction [Miné,06]

## Dreams

Hold fast to dreams  
For if dreams die  
Life is a broken-winged bird  
That cannot fly.

Hold fast to dreams  
For when dreams go  
Life is a barren field  
Frozen with snow.

Langston Hughes