CS 5430:
Information Flow
Part I: Static Enforcement

Fred B. Schneider
Samuel B Eckert Professor of Computer Science

Department of Computer Science
Cornell University
Ithaca, New York 14853
U.S.A.

Cornell CIS
Computer Science

Access Control

Access control associates restrictions with:

- Containers (CS5430)
  - access control lists, capabilities
- Values (CS5432)
  - information flow control

Example: $x := y;\ \ldots\ z := x$

- container: value in y can be leaked by reading z
- value: restrictions on z include all restrictions on y

  ... no need to trust clients who access y.

Flow-based Access Control (FBAC)

- Labels propagate with flow.
- Labels restrict allowed info flow.

Flow-Label Invariant (FLI):

$$v \rightarrow w \implies \Gamma(v) \sqsubseteq \Gamma(w)$$

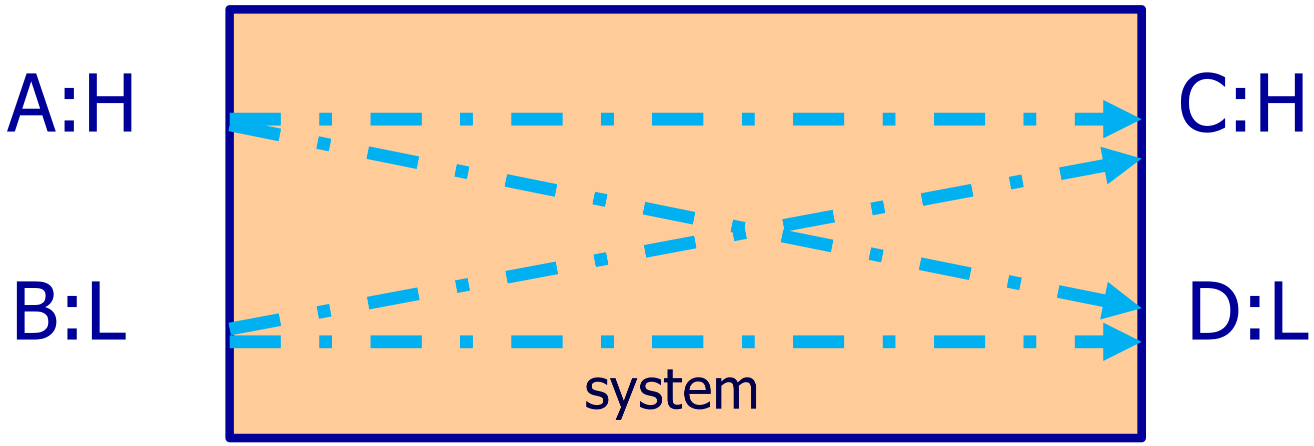
$v \rightarrow w$: v flows to w. *NB really* $\rightarrow_S$ *for flow in S*.

$\Gamma(v)$: label assoc with v --- gives restrictions on use of v

$\sqsubseteq$ reflexive and transitive relation on a set $\mathcal{L}$ of labels.

$\lambda_1 \sqsubseteq \lambda_2$: λ_2 includes <u>all</u> restrictions in λ_1

An application of FBAC

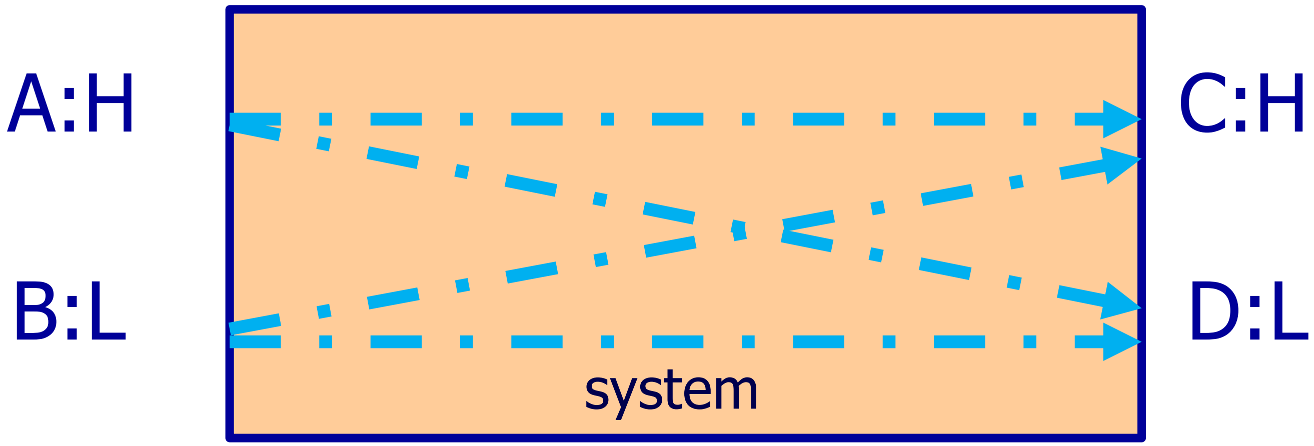


$$\mathcal{L} = \{ \mathrm{L}, \mathrm{H} \}$$

$$\mathrm{L} \sqsubseteq \mathrm{H}$$

$$\mathrm{v} \;\rightarrow\; \mathrm{w} \;\Longrightarrow\; \Gamma(\mathrm{v}) \sqsubseteq \Gamma(\mathrm{w})$$

An application of FBAC



$\mathcal{L} = \{ \mathrm{L}, \mathrm{H} \}$

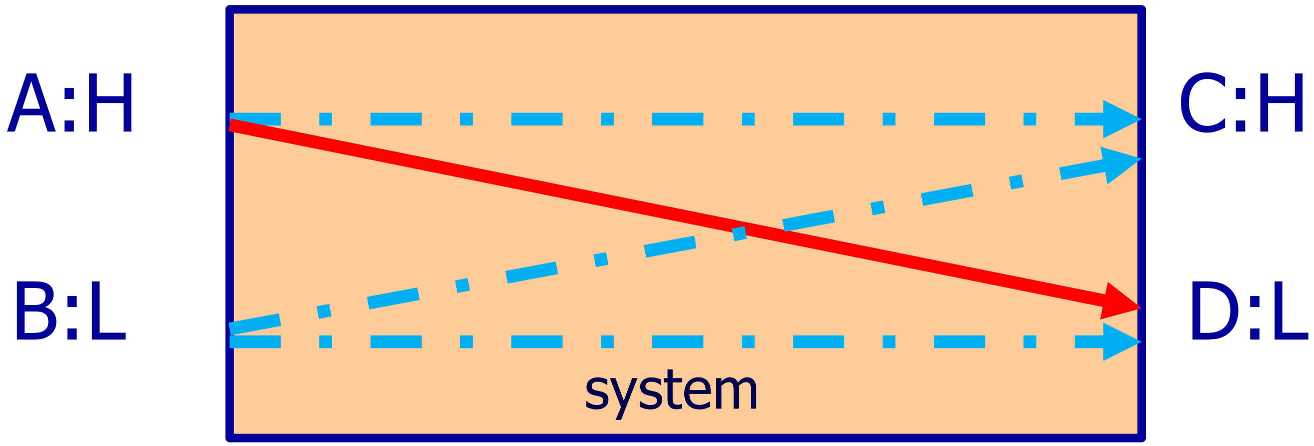
$\mathrm{L} \sqsubseteq \mathrm{H}$

$\mathrm{H} \not\sqsubseteq \mathrm{L}$

$$\mathrm{v} \rightarrow \mathrm{w} \implies \Gamma(\mathrm{v}) \sqsubseteq \Gamma(w)$$

$$= \Gamma(\mathrm{v}) \not\sqsubseteq \Gamma(\mathrm{w}) \implies \neg(\mathrm{v} \rightarrow \mathrm{w})$$

An application of FBAC

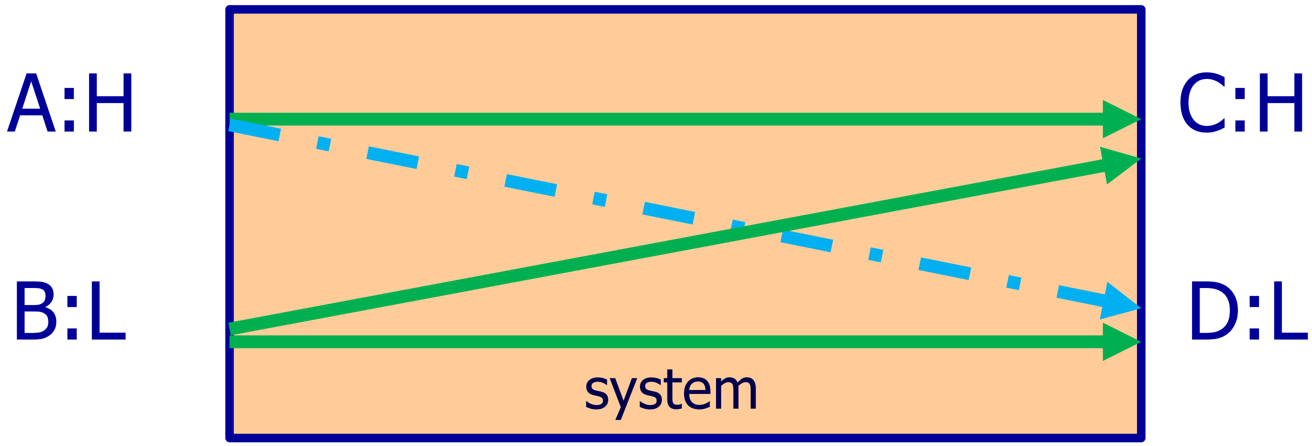


$$\mathcal{L} = \{\, \mathrm{L}, \mathrm{H} \,\}$$
$$\mathrm{L} \sqsubseteq \mathrm{H}$$
$$\mathrm{H} \not\sqsubseteq \mathrm{L}$$

$$\mathrm{v} \ \rightarrow \ \mathrm{w} \ \Longrightarrow \ \Gamma(\mathrm{v}) \sqsubseteq \Gamma(w)$$

$$= \Gamma(\mathrm{v}) \not\sqsubseteq \Gamma(\mathrm{w}) \ \Longrightarrow \neg(\mathrm{v} \ \rightarrow \ \mathrm{w})$$

An application of FBAC



$$\mathcal{L} = \{ \mathrm{L}, \mathrm{H} \}$$

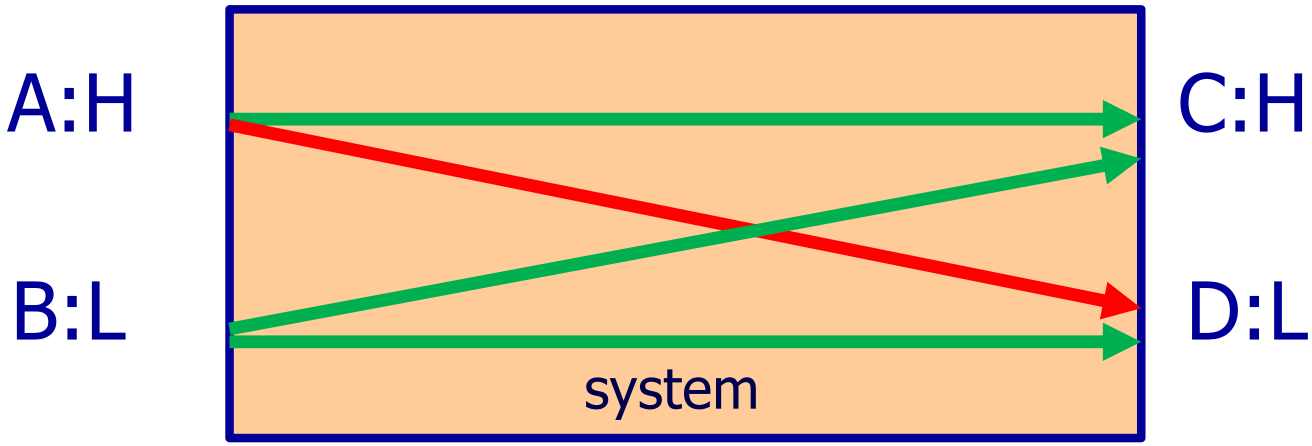
$$\mathrm{L} \sqsubseteq \mathrm{H}$$

$$\mathrm{H} \not\sqsubseteq \mathrm{L}$$

$$v \rightarrow w \implies \Gamma(v) \sqsubseteq \Gamma(w)$$

$$= \Gamma(v) \not\sqsubseteq \Gamma(w) \implies \neg(v \rightarrow w)$$

An application of FBAC



$$\mathcal{L} = \{ L, H \}$$
$$L \sqsubseteq H$$
$$H \not\sqsubseteq L$$

$$v \rightarrow w \implies \Gamma(v) \sqsubseteq \Gamma(w)$$
$$= \Gamma(v) \not\sqsubseteq \Gamma(w) \implies \neg(v \rightarrow w)$$

- <u>Confidentiality</u>: L: public and H: secret
- <u>Integrity</u>: L: trusted and H: untrusted

FBAC in General

Possible source/destination of flows:

- ports
- people
- variables

FBAC in Programs

Example: $x := y;$ … $z := x$

- $y \rightarrow x$, $x \rightarrow z$
- $\Gamma(y) \sqsubseteq \Gamma(x)$, $\Gamma(x) \sqsubseteq \Gamma(z)$.

FBAC in Programs

Example: $x := y; \ \ldots \ z := x$

- $y \rightarrow x$, $x \rightarrow z$
- $\Gamma(y) \sqsubseteq \Gamma(x)$, $\Gamma(x) \sqsubseteq \Gamma(z)$.
  - Conclude: If $y \rightarrow z$ then $\Gamma(y) \sqsubseteq \Gamma(z)$ also must hold.
  - Nb. $\rightarrow$ is not necessarily transitive.

Agenda

- Formalize Flow: $v \rightarrow v'$
  - Examples for intuition
  - Formal definitions
- Derive policies FBAC enforces:
  - Confidentiality
  - Integrity
- Means of enforcement
  - Static
  - Dynamic

$v \rightarrow v'$? Direct Flows in Programs

x := y mod 2

x := y * 0

z := y+2; x:= z

z := y+2; x := z-y

$v \rightarrow v'$? Direct Flows in Programs

$x := y \bmod 2$	$y \rightarrow x$
$x := y * 0$	$\neg (y \rightarrow x)$
$z := y+2;\ x := z$	$y \rightarrow x$
$z := y+2;\ x := z-y$	$\neg (y \rightarrow x)$

... Illustrates intransitive flow

$v \to v'$? Indirect Flows in Programs

if y>0 **then** x:=1 **else** x:=2 $y \to x$

while y>0 **do** x:=x+1; y:=y-1 **end** $y \to x$

Definitions for Flow

$v \rightarrow w$?

Satisfied if there exist two executions

- that differ only in the initial value of v –and-
- terminate having different final values of w.

$v \rightarrow w$? Formal Definition

Let

$\mathrm{dom}(m) = D$ for $m \in \mathrm{Mem}$

$[\![S]\!]$: $\mathrm{Mem} \longrightarrow \mathrm{Mem} \cup \{\bot\}$

$m =_V m'$: $(\forall v \in V\colon m(v) = m'(v))$

Define $v \rightarrow w$:

$$(\exists\, m, m'\colon\ m =_{D-\{v\}} m' \wedge [\![S]\!]m \neq \bot \wedge [\![S]\!]m' \neq \bot$$

$$\wedge\ [\![S]\!]m \neq_{\{w\}} [\![S]\!]m')$$

FBAC in action

Partition the set of all program variables: V_H and V_L

- $V_H = \{ v \mid \Gamma(v) = H\}$ $V_L = \{ v \mid \Gamma(v) = L\}$,
- $L \sqsubseteq H$.

For all $v_H \in V_H$, $v_L \in V_L$ FBAC requires

$$v_H \rightarrow v_L \Rightarrow \Gamma(v_H) \sqsubseteq \Gamma(v_L)$$

$$= v_H \rightarrow v_L \Rightarrow H \sqsubseteq L$$

$$= v_H \rightarrow v_L \Rightarrow \text{false}$$

$$= \neg (v_H \rightarrow v_L)$$

FBAC in action

$$\neg\,(v_H \rightarrow v_L)$$

$$= \neg(\,\exists\, m, m'\!:\;\; m =_{D-\{v\}} m' \;\wedge\; [\![S]\!]m \neq \perp \;\wedge\; [\![S]\!]m' \neq \perp$$
$$\wedge\; [\![S]\!]m \neq_{\{w\}} [\![S]\!]m')$$

$$= (\,\forall\, m, m'\!:\;\; m =_{D-\{v\}} m' \;\wedge\; [\![S]\!]m \neq \perp \;\wedge\; [\![S]\!]m' \neq \perp$$
$$\Rightarrow\; [\![S]\!]m =_{\{w\}} [\![S]\!]m')$$

Conclusion: Changes to v_H do not cause changes to v_L in terminating executions.

- Confidentiality: H is secret; L is public
- Integrity: H is untrusted; L is trusted.

Non-interference

Generalize variables v_H, v_L to sets V_H, V_L.

$$(\forall\, m, m':\ \ m =_{D-V_H} m' \wedge [\![S]\!]m \neq \perp \wedge [\![S]\!]m' \neq \perp$$
$$\Rightarrow\ \ [\![S]\!]m =_{V_L} [\![S]\!]m')$$

Changes to variables in V_H do not affect the final values of variables in V_L. Property (with terms often left implicit) is called :

- Termination **<u>in</u>**sensitive non-interference (TINI)
- Goguen-Meseguer non-interference
- Relational non-interference (RNI)

Additional Leaks: Termination

if $\mathrm{h} > 0$

  then while true **do skip end**

  else skip

fi

Termination leaks value of $\mathrm{h} > 0$.

Value of h flows to termination: $\mathrm{h} \to \perp$

$v \to \perp$? Formal Definition

$v \to \perp$:

$$(\exists\, m, m': \; m =_{D-\{v\}} m' \wedge ([\![S]\!]m = \perp) \neq ([\![S]\!]m' = \perp))$$

Define $\Gamma(\perp)$: Label needed by a principal in order to ascertain whether execution has terminated.

Usually $\Gamma(\perp) = L$.

Derive: Termination Sensitive NI 1/3

Flow-Label Invariant:

$$(v \to w \implies \Gamma(v) \sqsubseteq \Gamma(w)) \land (v \to \bot \implies \Gamma(v) \sqsubseteq \Gamma(\bot))$$

$$= (v \to w \implies \Gamma(v) \sqsubseteq \Gamma(w)) \land (v \to \bot \implies \Gamma(v) \sqsubseteq L)$$

$$= (\Gamma(v) = L)$$
$$\lor \ (\neg(v \to \bot) \ \land \ (v \to w \Rightarrow \Gamma(v) \sqsubseteq \Gamma(w)))$$

Termination Sensitive NI 2/3

$$\Gamma(v) = L \ \lor \ (\neg(v \to \bot) \ \land \ (v \to w \Rightarrow \Gamma(v) \sqsubseteq \Gamma(w)))$$

$\neg(v \to \bot)$:

$= \ \neg(\exists\, m, m' : \ m =_{D-\{v\}} m' \ \land ([\![S]\!]m = \bot) \neq ([\![S]\!]m' = \bot))$

$= \ (\forall\, m, m' : \ m =_{D-\{v\}} m' \ \Rightarrow ([\![S]\!]m = \bot) = ([\![S]\!]m' = \bot))$

$\neg(v \to w)$: $\{ \text{since } [\![S]\!]m \neq \bot \Rightarrow [\![S]\!]m' \neq \bot \}$

$= \neg(\exists\, m, m' : \ m =_{D-\{v\}} m' \ \land \ [\![S]\!]m \neq \bot \land ([\![S]\!]m \neq_{\{w\}} [\![S]\!]m'))$

$= (\forall\, m, m' : \ m =_{D-\{v\}} m' \ \land \ [\![S]\!]m \neq \bot \Rightarrow ([\![S]\!]m =_{\{w\}} [\![S]\!]m'))$

Termination Sensitive NI 3/3

Define: TSNI

$$(\forall\, m, m' : m =_{D-\{V_H\}} m' \Rightarrow$$
$$((\llbracket S \rrbracket m = \perp) = (\llbracket S \rrbracket m' = \perp))$$
$$\wedge\, (\llbracket S \rrbracket m \neq \perp \Rightarrow (\llbracket S \rrbracket m =_{\{V_L\}} \llbracket S \rrbracket m')))$$

Other Generalizations of $v \rightarrow w$

Let $\mathrm{dom}(m) = D$ for $m \in \mathrm{Mem}$

$[\![S]\!]$: $\mathrm{Mem} \longrightarrow \mathrm{Mem}^* \cup \{\perp\}$

$m =_V m'$: $(\forall v \in V\colon m(v) = m'(v))$

$(m_1 m_2 \ldots m_i \ \ldots) \approx_V (m'_1 m'_2 \ \ldots m'_i)$:

$(m_1|_V \, m_2|_V \ldots m_i|_V \ldots) =^* (m'_1|_V \, m'_2|_V \ldots m'_i|_V)$

where: $=^*$ is equality of de-stuttered sequences.

Define $v \rightarrow w$:

$(\exists\, m, m'\colon\ m =_{D-\{v\}} m' \wedge [\![S]\!]m \neq \perp \wedge [\![S]\!]m' \neq \perp$

$\wedge \neg ([\![S]\!]m \approx_{\{w\}} [\![S]\!]m'))$

Enforcement of FBAC

FLI potentially imposes restrictions on statements.

- Static Enforcement
  - Compiler ensures program is type correct.
  - Type correct programs will satisfy restrictions.
- Dynamic Enforcement
  - Insert run-time checks that halt program execution about to violate restrictions.
  - Change labels to satisfy restrictions as program execution proceeds.

Toy Language

e ::= x | n | e1+e2 | …

c ::= x := e

| **if** e **then** c1 **else** c2 **fi**

| **while** e **do** c **end**

| c1; c2

Restrictions for:

Assignment $x := e$

$$v \to w \implies \Gamma(v) \sqsubseteq \Gamma(w)$$

$x := y$ causes $y \to x$

- requires $\Gamma(y) \sqsubseteq \Gamma(x)$

$x := y+z$ causes $y \to x$ and $z \to x$

- requires: $\Gamma(y+z) \sqsubseteq \Gamma(x)$
- implied by: $\Gamma(y) \sqcup \Gamma(z) \sqsubseteq \Gamma(x)$

Restrictions for:
Assignment $x := E$

$x := E$ causes $E \rightarrow x$

define $E \rightarrow x$: $(\forall\, v \in E\colon\ v \rightarrow x)$

define $\Gamma(E)$: $(\sqcup\, \Gamma(v) \in E)$

where $\lambda \sqcup \lambda'$ is smallest label satisfying

$\lambda \sqsubseteq \lambda \sqcup \lambda'$ and $\lambda' \sqsubseteq \lambda \sqcup \lambda'$

Restrictions for:

Assignment $x := E$

$x := E$ causes $E \rightarrow x$

define $E \rightarrow x$: $(\forall\, v \in E:\ v \rightarrow x)$

define $\Gamma(E)$: $(\sqcup\, \Gamma(v) \in E)$

where $\lambda \sqcup \lambda'$ is smallest label satisfying

$\lambda \sqsubseteq \lambda \sqcup \lambda'$ and $\lambda' \sqsubseteq \lambda \sqcup \lambda'$

$x := E$ causes $E \rightarrow x$

– requires $(\sqcup\, \Gamma(v) \in E) \sqsubseteq \Gamma(x)$

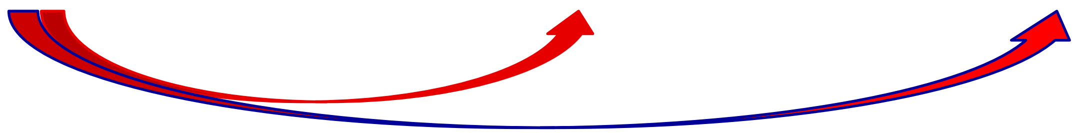
Restrictions for:

If Statements

$$\textbf{if}\ \ y > 0\ \ \textbf{then}\ \ x := 1\ \ \textbf{else}\ \ x := 2\ \ \textbf{fi}$$

Restrictions for:
If Statements

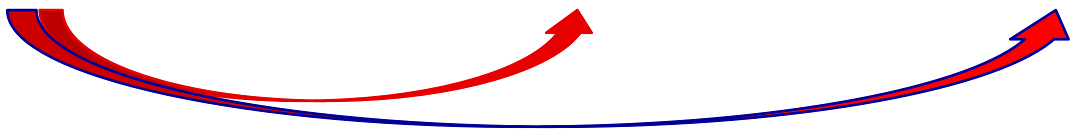
$$\textbf{if}\ \ y > 0\ \ \textbf{then}\ \ x := 1\ \ \textbf{else}\ \ x := 2\ \ \textbf{fi}$$



Restrictions for:

If Statements

if $y > 0$ **then** $x := 1$ **else** $x := 2$ **fi**



$y > 0 \rightarrow pc, \quad pc \rightarrow x,$

Restrictions for:

If Statements

$$\textbf{if}\ \ y > 0\ \ \textbf{then}\ \ x := 1\ \ \textbf{else}\ \ x := 2\ \ \textbf{fi}$$

$y > 0 \rightarrow pc,\quad pc \rightarrow x,\ \ y > 0 \rightarrow x$

$y > 0 \rightarrow x$ requires $\Gamma(y > 0) \sqsubseteq \Gamma(x)$

Restrictions for:

If Statements

$$\textbf{if } y > 0 \ \textbf{ then } x := 1 \ \textbf{ else } \ x := 2 \ \textbf{ fi}$$

$$\begin{aligned} ctx &= \Gamma(y > 0) \\ &= \Gamma(y) \sqcup \Gamma(0) \\ &= \Gamma(y) \end{aligned}$$

Restrictions for:

If Statements

$$\textbf{if } \mathrm{B} \ \textbf{ then } \mathrm{x} := \mathrm{E} \ \textbf{ else } \ \ldots \ \textbf{ fi}$$

$$B \rightarrow x, \qquad E \rightarrow x$$

Restrictions for:

If Statements

$$\textbf{if}\ \ \mathrm{B}\ \ \textbf{then}\ \ \mathrm{x} \coloneqq \mathrm{E}\ \ \textbf{else}\ \ \ldots\ \ \textbf{fi}$$

$$B \to x, \qquad E \to x$$

requires: $\Gamma(B) \sqsubseteq \Gamma(x),\ \Gamma(E) \sqsubseteq \Gamma(x)$

Restrictions for:
If Statements

$$\textbf{if} \;\; \mathrm{B} \;\; \textbf{then} \;\; \mathrm{x} := \mathrm{E} \;\; \textbf{else} \;\; \ldots \;\; \textbf{fi}$$

$$B \rightarrow x, \qquad E \rightarrow x$$

requires: $\Gamma(B) \sqsubseteq \Gamma(x),\ \Gamma(E) \sqsubseteq \Gamma(x)$

implied by:

$$\mathrm{ctx} = \Gamma(B)$$

$$\mathrm{ctx} \sqcup \Gamma(\mathrm{E}) \sqsubseteq \Gamma(\mathrm{x})$$

Restrictions for: Nested If Statements

```
if z>0
        then  y := 23
              if y> 0
                    then  x:= 1
                    else  u:= 2
              fi
        else
              w:=3
fi
```

Restrictions for: Nested If Statements

```
if z>0
    then   y := 23
           if y> 0
               then x:= 1 ---  ctx = Γ(y)
               else u:= 2 ---  ctx = Γ(y)
           fi
    else
           w:=3
fi
```

Restrictions for: Nested if Statements

```
if z>0
    then  y := 23 --- ctx = Γ(z)
          if y> 0
              then  x:= 1 --- ctx = Γ(y) ⊔ Γ(z)
              else  u:= 2 --- ctx = Γ(y) ⊔ Γ(z)
          fi
    else
          w:=3  --- ctx = Γ(z)
fi
```

A Type System

- Fixed label assignment Γ
- Goal:
  - Type correctness implies Flow-Label invariant will hold throughout executions.

    $$v \rightarrow w \implies \Gamma(v) \sqsubseteq \Gamma(w)$$

  - Flow-Label invariant implies RNI will hold throughout executions.

Type Systems: Big Picture

"Program S is type correct"   is a theorem in a logic (say) secL.

- Logic is decidable.
  - Compiler's type checker "proves" these theorems.
- Logic is sound with respect to:

  "Program S satisfies FLI invariant"

Formulas of secL

Formulas of secL are called judgements.

Formulas of secL are given as sequents:

- $\Gamma, ctx \vdash Expr : \lambda$ for expression *Expr*, label λ

  $\Gamma, ctx \vdash S$ for statement S

Inference rules give premises and conclusion

$$\frac{P_1, P_2 \,, \ldots \,, \; P_n}{C}$$

Rules for Expressions

- Constant: $$\frac{}{\Gamma, \text{ctx} \vdash n : L}$$

- Variable: $$\frac{\Gamma(x) = \lambda}{\Gamma, \text{ctx} \vdash x : \lambda}$$

- Expression: $$\frac{\Gamma, \text{ctx} \vdash e : \lambda \quad \Gamma, \text{ctx} \vdash e' : \lambda'}{\Gamma, \text{ctx} \vdash e + e' : \lambda \sqcup \lambda'}$$

A Proof (1/3)

Given $\Gamma(x) = L$ and $\Gamma(y) = H$:

$$\frac{\Gamma(x) = L}{\Gamma, ctx \vdash x : L}$$

A Proof (2/3)

Given $\Gamma(x) = L$ and $\Gamma(y) = H$:

$$\frac{\Gamma(x) = L}{\Gamma, ctx \vdash x : L} \quad \frac{\Gamma(y) = H}{\Gamma, ctx \vdash y : H}$$

A Proof (3/3)

Given $\Gamma(x) = L$ and $\Gamma(y) = H$:

$$\dfrac{\dfrac{\Gamma(x) = L}{\Gamma, ctx \vdash x : L} \quad \dfrac{\Gamma(y) = H}{\Gamma, ctx \vdash y : H}}{\Gamma, ctx \vdash x + y : L \sqcup H}$$

Conclusion: $x{+}y : H$ $\quad$ (since $L \sqcup H = H$)

Assignment Rule

$x := E$

- causes: $E \rightarrow x$
- requires: $\Gamma(E) \sqsubseteq \Gamma(x)$

Assign: $$\frac{\Gamma, ctx \vdash E: \lambda \ , \ \ \lambda \sqcup ctx \sqsubseteq \Gamma(x)}{\Gamma, ctx \vdash x := E}$$

if Rule

$$\frac{\Gamma, \mathrm{ctx} \vdash \mathrm{e} : \lambda, \quad \Gamma, \lambda \sqcup \mathrm{ctx} \vdash \mathrm{C}_1, \quad \Gamma, \lambda \sqcup \mathrm{ctx} \vdash \mathrm{C}_2}{\Gamma, \mathrm{ctx} \vdash \textbf{if}\ \mathrm{e}\ \textbf{then}\ \mathrm{C}_1\ \textbf{else}\ \mathrm{C}_2\ \textbf{fi}}$$

if Rule Example Proof

1. Constant:

$$\frac{}{\Gamma, (L \sqcup ctx) \vdash 1{:}L}$$

2. Assign:

$$\frac{\Gamma, (L \sqcup ctx) \vdash 1{:}L, \quad L \sqcup (L \sqcup ctx) \sqsubseteq \Gamma(x)}{\Gamma, (L \sqcup ctx) \vdash x{:=}1}$$

3. if

$$\frac{\Gamma, ctx \vdash y>0 : L \quad \Gamma, L \sqcup ctx \vdash x{:=}1 \quad \Gamma, L \sqcup ctx \vdash x{:=}2}{\Gamma, ctx \vdash \text{if } y>0 \text{ then } x{:=}1 \text{ else } x{:=}2 \text{ fi}}$$

while Rule

while:

$$\frac{\Gamma, \mathrm{ctx} \vdash \mathrm{e} : \lambda \qquad \Gamma, \lambda \sqcup \mathrm{ctx} \vdash \mathrm{C}}{\Gamma, \mathrm{ctx} \vdash \textbf{while}\ \mathrm{e}\ \textbf{do}\ \mathrm{c}\ \textbf{end}}$$

; (sequence) rule

$$\frac{\Gamma, \mathrm{ctx} \vdash \mathrm{C}_1, \qquad \Gamma, \mathrm{ctx} \vdash \mathrm{C}_2}{\Gamma, \mathrm{ctx} \vdash \mathrm{C}_1 \;; \mathrm{C}_2}$$

secL Type System Retrospective

- Soundness
  - Type correct programs satisfy
    - Flow-Label Invariant: $v \rightarrow w \implies \Gamma(v) \sqsubseteq \Gamma(w)$
    - Relational non-interference (RNI)
  - If program doesn't satisfy
    - Flow-Label invariant or
    - RNI

  then program won't be type correct.

secL Type System Retrospective

- (in)Completeness
  - The type system is incomplete.
  - If a program is not type correct then that program might still satisfy Flow-Label invariant and RNI.

– $$\frac{\Gamma,ctx \vdash y*0 : H, \quad H \sqcup L \sqsubseteq \Gamma(x)}{\Gamma, L \vdash x := y*0}$$

If $\Gamma(x) = L$ ...

- Type checking fails

secL Type System Retrospective

- (in)Completeness
  - The type system is incomplete.
  - If a program is not type correct then that program might still satisfy Flow-Label invariant and RNI.

– $$\frac{\Gamma,\mathrm{ctx} \vdash y * 0 : H, \quad H \sqcup L \sqsubseteq \Gamma(x)}{\Gamma, L \vdash x := y * 0}$$

If $\Gamma(x) = L$ ...

- Type checking fails
- FLI invariant valid
- RNI satisfied.

Eliminate Incompleteness?

Sequence rule

$$\frac{\Gamma, \mathrm{ctx} \vdash \mathrm{C}_1, \qquad \Gamma, \mathrm{ctx} \vdash \mathrm{C}_2}{\Gamma, \mathrm{ctx} \vdash \mathrm{C}_1 \;;\; \mathrm{C}_2}$$

Consider:

$$\mathbf{if}\ \mathrm{h} > 0\ \mathbf{then}\ C;\ \mathrm{v}_{\mathrm{L}} := 2\ \mathbf{else\ skip\ fi}$$

- Satisfies RNI (=termination insensitive) if C diverges.
- Sequence rule must predict that C_1 diverges.
  - Predicting divergence requires solving the halting problem.

Program with Termination Channel

$$\textbf{while } v_H > 0 \textbf{ do skip end}; \;\; v_L := 2$$

- Program is secL type correct.
- Program satisfies RNI.
- Program does not satisfy termination sensitive non interference (TSNI): $v_H \rightarrow \perp$

Type system for TSNI

Prevent channel arising from infinite loops.

- Allow only L terms in **while** guards.
  - Loop termination does not depend of H values.

$$\frac{\Gamma,\mathrm{ctx} \vdash e:L \qquad \Gamma,\mathrm{ctx} \vdash C}{\Gamma,\mathrm{ctx} \vdash \textbf{while}\ e\ \textbf{do}\ C\ \textbf{end}}$$

- Type correct programs now exhibit TSNI.
- What about loops involving H terms?