

Parte I: Un Sistema di Tipi per la Sicurezza del Flusso delle Informazioni

Vedi: **Volpano-Smith-Irvine**

www.cs.unibo.it/~laneve/html/metodi1/paper1.ps

Il Problema:

in un programma i dati sono partizionati in diversi “livelli di sicurezza”.
Bisogna verificare che questi livelli non siano violati, ad esempio copiando una variabile ad un livello di sicurezza in una variabile ad un livello di sicurezza più basso. (**soundness**)

Esempio:

	<i>variabili</i>
livello 1 :	x,y,z
livello 0:	a,b,c

x := y

a := b

x := a

a := x



Proprietà: il flusso dell'informazione è consentito solamente da un livello più basso ad un livello più alto.

Cf. "non-interferenza" tra dati di diversi utenti [Goguen-Meseguer]

Un Semplice Linguaggio Strutturato a Blocchi

Il linguaggio consiste di *comandi* C ed *espressioni* E :

C	$::= E := E'$	(assegnamento)
	$C ; C'$	(comp. sequenziale)
	$\text{if } E \text{ then } C \text{ else } C'$	(condizionale)
	$\text{while } E \text{ do } C$	(iterazione)
	$\text{let } x := E \text{ in } C$	(blocco)
E	$::= x$	(variabili)
	Z	(numeri interi)
	$(E + E')$	
	$(E - E')$	
	$(E = E')$	
	$(E < E')$	

Osservazione: • x rappresenta un generico *identificatore*. Il programma può introdurre nuovi identificatori (vedi "blocco"). Le variabili introdotte sono dette "*legate*", le altre sono dette "*libere*"

n rappresenta un generico *intero*, che sono i soli *valori* (0 rappresenta "falso", $\neq 0$ rappresenta "vero")

Osservazione: Non ci sono primitive di I/O. L' I/O è fatto attraverso le variabili libere:

per l'input si legge una variabile libera;

per l'output si scrive in una variabile libera.

Esempi: Si assuma che h sia una variabile "sicura" ed l sia una variabile "insicura"

flusso esplicito da h ad l :

```
 $l := (h + 1) ;$ 
```

il valore finale di l è il valore iniziale di h più 1.

```
 $\text{if } (h > l) \text{ then } l := 0 \text{ else } l := 1 ;$ 
```

il valore finale di l dipende dal fatto che esso sia più piccolo o più grande di h .

flusso implicito da h ad l:

```
while (h > 0) do
  l := (l + 1) ;
  h := (h - 1) ;
```

il valore finale di l è la somma dei valori iniziali di l ed h.

```
if (h > 0) then let l := 1 in C
  else let l := 0 in C' ;
```

il valore di l è vero o falso a seconda se h è positivo o negativo. Dunque C e C', attraverso una variabile "insicura", hanno informazioni di una variabile "sicura"

Osservazione: tutti questi programmi sono *crepe della sicurezza del flusso delle informazioni (secure flow leaks)*

Le regole per i Tipi per la Sicurezza del Flusso delle Informazioni

tipi: $T ::=$

- | *L-int*
- | *H-int*
- | *L-int var*
- | *H-int var*
- | *L-cmd*
- | *H-cmd*

ordinamento: (set-based interpretation)

- *L-int* < *H-int*
- *H-cmd* < *L-cmd*

Osservazione: non c'è alcuna relazione per tipi "*L/H-int var*"

Commenti:

H-cmd < L-cmd

dice che l'insieme dei comandi di una certa classe di sicurezza sono un sottoinsieme di quelli ad un livello inferiore.

i tipi "*L-int var*" e "*H-int var*" non sono ordinati perchè la classe di sicurezza di una variabile non può essere modificato.

una espressione di una certa classe di sicurezza può essere considerata di livello superiore. Questo consente il flusso di informazione dal basso verso l'alto ma non viceversa (cf. assegnamento).

un altro modo per presentare l'ordinamento è il seguente:

- ci sono livelli di sicurezza che sono un ordinamento parziale. Nel nostro caso questo ordinamento parziale è $L < H$.
- sulla base di un ordinamento di livelli di sicurezza, si definiscono:
 - (monotonia, per le espressioni) se $l < l'$ allora $-int < '-int$
 - (antimonotonia, per i comandi) se $l < l'$ allora $'-cmd < -cmd$

Il formato delle asserzioni (judgments):

$\Gamma \vdash C : H\text{-cmd}$	(per i comandi)
$\Gamma \vdash C : L\text{-cmd}$	
$\Gamma \vdash E : H\text{-int var}$	(per le variabili)
$\Gamma \vdash E : L\text{-int var}$	
$\Gamma \vdash E : H\text{-int}$	(per le altre espressioni)
$\Gamma \vdash E : L\text{-int}$	

dove l' *ambiente* Γ è il *tipo degli identificatori*

L'asserzione significa che i codici C ed E hanno rispettivamente tipo $H\text{-cmd}$, $L\text{-cmd}$, $H\text{-int var}$, $L\text{-int var}$, oppure $H\text{-int}$, $L\text{-int}$, quando le variabili hanno tipo Γ . Più brevemente:

$\Gamma \vdash P :$ si legge "P è *ben-tipato* con tipo Γ in ambiente Γ "

L'ambiente Γ

Γ è una *funzione parziale a dominio finito* dall'insieme delle variabili ai tipi.

Esempio:

$$\Gamma : \begin{array}{ll} a & L\text{-int var} \\ x & H\text{-int var} \end{array}$$

e $\Gamma(x) = H\text{-int var}$.

La notazione per *modificare* una funzione è

$$\Gamma [x : L\text{-int var}]$$

Esempio:

$$\Gamma [x : L\text{-int var}] : \begin{array}{ll} a & L\text{-int var} \\ x & L\text{-int var} \end{array}$$

Il formato delle regole: (due tipi)

assiomi

$\Gamma \vdash p :$

un certo codice p ha tipo τ in Γ

Ad esempio, $\Gamma \vdash 5 : L\text{-int}$

significa che, qualunque sia Γ (anche vuoto), 5 ha tipo " $L\text{-int}$ ".

regole di inferenza

$$\frac{\Gamma \vdash p_1 : \tau_1 \quad \dots \quad \Gamma \vdash p_k : \tau_k}{\Gamma \vdash p : \tau}$$

Se i codici p_1, \dots, p_k hanno tipo τ_1, \dots, τ_k in Γ , allora il codice p ha tipo τ in Γ

Esempio:

$$\frac{\Gamma \vdash \mathbf{E} : L\text{-int} \quad \Gamma \vdash \mathbf{E}' : L\text{-int}}{\Gamma \vdash (\mathbf{E} + \mathbf{E}') : L\text{-int}}$$

significa che, se gli addendi di una somma hanno tipo low-integer in Γ , anche la somma ha lo stesso tipo in Γ .

Cf. *Insiemi e relazioni induttive*.

Regole per le espressioni: (int = H, L)

(int) $\Gamma \vdash n : L\text{-int}$ $\Gamma \vdash n : H\text{-int}$

(ide)
$$\frac{\Gamma(\mathbf{x}) = \text{-int var}}{\Gamma \vdash \mathbf{x} : \text{-int var}}$$

(expr#)
$$\frac{\Gamma \vdash \mathbf{E} : \text{-int} \quad \Gamma \vdash \mathbf{E}' : \text{-int}}{\Gamma \vdash (\mathbf{E} \# \mathbf{E}') : \text{-int}} \quad \# \quad \{+, -, =, <\}$$

(r-val)
$$\frac{\Gamma \vdash \mathbf{E} : \text{'-int var} \quad \text{'-int} \leq \text{-int}}{\Gamma \vdash \mathbf{E} : \text{-int}}$$

Commenti

1. in (r-val) si ammette che una espressione di tipo `'-int var --` una variabile `--` possa avere un "tipo più sicuro" quando considerata come valore. Questa "*coercion*" è proibita altrimenti.
2. nelle regole (int) ed (expr#) si richiede che il tipo non sia `"-int var"`. In altri termini, i valori right-hand-side non possono essere `"-int var"`.

Regole per i comandi:

$$\text{(assign)} \quad \frac{\Gamma \vdash E : \text{'-int var} \quad \Gamma \vdash E' : \text{'-int} \quad \text{'-cmd} \leq \text{-cmd}}{\Gamma \vdash E := E' : \text{-cmd}}$$

$$\text{(compose)} \quad \frac{\Gamma \vdash c : \text{-cmd} \quad \Gamma \vdash c' : \text{-cmd}}{\Gamma \vdash c ; c' : \text{-cmd}}$$

$$\text{(if)} \quad \frac{\Gamma \vdash E : \text{'-int} \quad \Gamma \vdash c : \text{'-cmd} \quad \Gamma \vdash c' : \text{'-cmd} \quad \text{'-cmd} \leq \text{-cmd}}{\Gamma \vdash \text{if } E \text{ then } c \text{ else } c' : \text{-cmd}}$$

$$\text{(while)} \quad \frac{\Gamma \vdash E : \text{'-int} \quad \Gamma \vdash c : \text{'-cmd} \quad \text{'-cmd} \leq \text{-cmd}}{\Gamma \vdash \text{while } E \text{ do } c : \text{-cmd}}$$

$$\text{(let)} \frac{\Gamma \vdash E : \text{-int} \quad \Gamma[x : \text{-int var}] \vdash C : \text{'-cmd}}{\Gamma \vdash \text{let } x := E \text{ in } C : \text{'-cmd}}$$

Commenti (continua)

3. (assign) + le regole per le espressioni, consentono di assegnare un valore "meno sicuro" ad una variabile "più sicura"
4. in (let), non c'è alcuna relazione tra il tipo del comando C e quello dell'espressione E
5. in tutte le regole la sicurezza decresce o resta uguale

Osservazione: le regole riflettono la struttura sintattica dei termini. C'è esattamente una regola per ogni produzione sintattica.

Esempi: Sia $\Gamma = [a : L\text{-int var}, x : H\text{-int var}]$

1. $x := a.$ (r-val) implementa il flusso basso alto

$$\begin{array}{c}
 \Gamma \vdash a : L\text{-int var} \quad L\text{-int} \leq H\text{-int} \\
 \text{(r-val)} \text{-----} \\
 \Gamma \vdash a : H\text{-int} \\
 \Gamma \vdash x : H\text{-int var} \quad \Gamma \vdash a : H\text{-int} \\
 \text{(assign)} \text{-----} \\
 \Gamma \vdash x := a : H\text{-cmd}
 \end{array}$$

2. $a := x.$

$$\begin{array}{c}
 \Gamma(a) = H\text{-int var} \quad \text{💣} \\
 \text{(ide)} \text{-----} \\
 \Gamma \vdash a : H\text{-int var} \quad \Gamma \vdash x : H\text{-int} \quad H\text{-cmd} \leq \text{-cmd} \\
 \text{(assign)} \text{-----} \\
 \Gamma \vdash a := x : \text{-cmd}
 \end{array}$$

3. `while (x > 0) do (a := (a + 1) ; x := (x - 1)) ;`

$$\begin{array}{c}
 \Gamma (a) = H\text{-int var} \text{💣} \\
 \text{(ide)} \frac{}{\Gamma \vdash a : H\text{-int var} \quad \Gamma \vdash (a - 1) : H\text{-int} \quad H\text{-int} \leq H\text{-int}} \\
 \text{(assign)} \frac{}{\Gamma \vdash a := (a + 1) : H\text{-cmd}} \\
 \text{.....} \\
 \text{(assign)} \frac{}{\Gamma \vdash x := (x - 1) : H\text{-cmd}} \\
 \text{(compose)} \frac{}{\Gamma \vdash a := (a + 1) ; x := (x - 1) : H\text{-cmd}} \\
 \text{.....} \\
 \text{(expr-)} \frac{}{\Gamma \vdash (x > 0) : H\text{-int}} \\
 \text{(while)} \frac{}{\Gamma \vdash \text{while } (x > 0) \text{ do } (a := (a + 1) ; x := (x - 1)) : H\text{-cmd}}
 \end{array}$$

Esempio: Il Massimo Comune Divisore.

Sia

$\Gamma = [l_{in} : L\text{-int var}, l_{in}' : H\text{-int var}, l_{out} : H\text{-int var}]$

[si assuma che $l_{in} > l_{in}'$]

let $x := l_{in}$ in let $y := l_{in}'$ in

$C \left\{ \begin{array}{l} \text{while } x > 0 \text{ do} \\ \quad x := x - y ; \\ \quad \text{if } y > x \text{ then let } z := x \\ \qquad \qquad \qquad \text{in } x := y ; y := z ; \\ \quad \text{else } x := 0 ; \end{array} \right. \right\} C'$

$l_{out} := y ;$

Siano

$\Gamma' = \Gamma [x : H\text{-int var}] ; \quad \Gamma'' = \Gamma' [y : H\text{-int var}]$

$$\begin{array}{c}
\Gamma \vdash l_{in} : L\text{-int var} \\
L\text{-int} \leq H\text{-int} \\
\text{(r-val)} \frac{\quad}{\Gamma \vdash l_{in} : H\text{-int}}
\end{array}
\quad
\begin{array}{c}
\Gamma' \vdash l_{in}' : H\text{-int var} \\
H\text{-int} \leq H\text{-int} \\
\text{(r-val)} \frac{\quad}{\Gamma' \vdash l_{in}' : H\text{-int}}
\end{array}
\quad
\begin{array}{c}
\text{(1)} \\
\frac{\quad}{\Gamma'' \vdash C ; l_{out} := y : H\text{-cmd}}
\end{array}$$

$$\Gamma \vdash \text{let } x := l_{in} \text{ in let } y := l_{in}' \text{ in } C ; l_{out} := y : H\text{-cmd}$$

La prova di (1)

$$\begin{array}{c} \text{(2)} \\ \frac{\Gamma'' \vdash C' : H\text{-cmd}}{\text{(while)} \quad \frac{\Gamma'' \vdash C' : H\text{-cmd} \quad \frac{\text{(expr>)} \quad \frac{\dots\dots\dots}{\Gamma'' \vdash x>0 : H\text{-int}}{\Gamma'' \vdash \text{while } x>0 \text{ do } C' : H\text{-cmd}}}{\Gamma'' \vdash \text{while } x>0 \text{ do } C' : H\text{-cmd}}} \\ \dots\dots\dots \\ \frac{\Gamma'' \vdash l_{\text{out}} := y : H\text{-cmd}}{\text{(compose)} \quad \frac{\Gamma'' \vdash l_{\text{out}} := y : H\text{-cmd} \quad \Gamma'' \vdash \text{while } x>0 \text{ do } C' : H\text{-cmd}}{\Gamma'' \vdash \text{while } x>0 \text{ do } C' ; l_{\text{out}} := y : H\text{-cmd}}} \end{array}$$

La prova di (2) $\Gamma''' = \Gamma'' [z : H-int var]$

$$\begin{array}{c}
 \dots\dots\dots \\
 \text{(let)} \frac{\Gamma''' \vdash x : H-int \quad \Gamma''' \vdash x := y ; y := z : H-cmd}{\Gamma'' \vdash \text{let } z := x \text{ in } x := y ; y := z : H-cmd} \\
 \text{(expr>)} \frac{\dots\dots\dots}{\Gamma \vdash y > x : H-int} \quad \Gamma \vdash x := 0 : H-cmd \\
 \text{(if)} \frac{\Gamma \vdash y > x : H-int \quad \Gamma \vdash x := 0 : H-cmd}{\Gamma \vdash \text{if } y > x \text{ then let ... else } x := 0 : H-cmd} \\
 \text{(assign)} \frac{\dots\dots\dots}{\Gamma \vdash x := x-y : H-cmd} \\
 \text{(compose)} \frac{\Gamma \vdash x := x-y : H-cmd}{\Gamma \vdash C' : H-cmd}
 \end{array}$$

La tecnica di prova standard per dimostrare proprietà sarà l'induzione strutturale.

Come esempio di applicazione, verifichiamo la seguente proprietà:

Lemma di Weakening. Se $\Gamma \vdash P$ e $x \notin \text{dom}(\Gamma)$ allora $\Gamma[x : \text{int-var}] \vdash P$.

Prova: Per induzione sulla struttura dell'albero di prova di $\Gamma \vdash P$.

(caso di base) In questo caso l'albero è un nodo.

Le uniche regole che possono produrre un tale albero sono **(int)** ed **(ide)**.

Verifichiamo la seconda. Sia $\Gamma \vdash y : \text{' var}$ e $x \notin \text{dom}(\Gamma)$. Il judgment $\Gamma \vdash y : \text{' var}$ è derivato da **(ide)** con premessa $\Gamma(y) = \text{' var}$. Dunque $\Gamma[x : \text{var}] \vdash (y) = \text{' var}$, poichè $x \notin \text{dom}(\Gamma)$.

Applicando **(ide)** a quest'ultimo judgment si ottiene $\Gamma[x : \text{var}] \vdash y : \text{' var}$.

(caso induttivo) Supponiamo che il lemma valga per alberi di una certa struttura e dimostriamolo per alberi di struttura più complessa.

Ragioniamo sull'ultima regola (quella che produce la radice dell'albero).

I casi possibili sono (*expr#*), (*r-val*), (*assign*), (*compose*), (*if*), (*while*) e (*let*).

Analizziamo il caso di (*let*), gli altri sono più semplici. In questo caso l'albero di prova termina così:

$$\text{(let)} \quad \frac{\Gamma \vdash E : ' \quad \Gamma[y : ' var] \vdash C : " cmd}{\Gamma \vdash \text{let } y := E \text{ in } C : " cmd}$$

Per ipotesi $x \in \text{dom}(\Gamma)$. Per ipotesi induttiva applicata a $\Gamma \vdash E : '$, otteniamo $\Gamma[x : var] \vdash E : ' (1)$.

Per quanto riguarda $\Gamma[y : ' var] \vdash C : " cmd$, ci sono due casi: $x = y$ e $x \neq y$.

Nel primo caso $\Gamma[x : \text{var}] [y : ' \text{var}] \vdash \Gamma[y : ' \text{var}]$. Quindi $\Gamma[x : \text{var}] [y : ' \text{var}] \vdash C : \text{cmd}$ (2).

Da (1) e (2), con la regola (let), si deduce $\Gamma[x : \text{var}] \vdash \text{let } y := E \text{ in } C : \text{cmd}$.

Se $x \neq y$, per ipotesi induttiva su $\Gamma[y : ' \text{var}] \vdash C : \text{cmd}$, deriviamo $\Gamma[x : \text{var}] [y : ' \text{var}] \vdash C : \text{cmd}$ (3).

Come prima, il lemma segue da (1) e (3) con (let).

Le Proprietà del Sistema di Tipi.

Le proprietà di base del verificatore:

Monotonia -- Lemma 1.

Se una espressione è ben tipata a livello di sicurezza $-int$ allora essa è ben tipata a livello $'-int$, con $' \leq$.

Se un comando è ben tipato a livello di sicurezza $-cmd$ allora esso è ben tipato a livello $'-cmd$, con $' \leq$.

Sicurezza -- Lemma 2. *Un'espressione di una certa classe di sicurezza contiene soltanto variabili di classe di sicurezza uguale o inferiore.*

Integrità -- Lemma 3. *Ogni assegnamento in un comando ben-tipato in una certa classe di sicurezza modifica variabili di classe uguale o superiore.*

Lemma 1 (Monotonia). Se $\Gamma \vdash C : -cmd$ e $-cmd \leq '-cmd$ allora $\Gamma \vdash C : '-cmd$. Se $\Gamma \vdash E : -int$ e $-int \leq '-int$ allora $\Gamma \vdash E : '-int$.

Prova: per induzione sull'altezza degli alberi di prova di $\Gamma \vdash C : cmd$ e $\Gamma \vdash E : int$.

Caso di Base. L'unica possibilità è che l'albero sia istanza di (int) e (ide). Nel primo caso la regola (int) consente di derivare la conclusione $\Gamma \vdash n : 'int$.

(ide) vuoto (l'ipotesi non si applica).

Caso Induttivo. Questo caso contiene tre sottocasi per le espressioni e cinque per i comandi, a seconda che l'ultima regola applicata nella dimostrazione sia istanza di (expr#), (r-val) oppure di (assign), (compose), (if), (while), (let).

(expr#) Per ipotesi induttiva, devono valere $\Gamma \vdash E : '-int$ e $\Gamma \vdash E' : '-int$.
Dunque, attraverso (expr#) otteniamo $\Gamma \vdash E \# E' : '-int$.

(r-val) Per ipotesi, deve valere $\Gamma \vdash E : ''-int var$ e $''-int \leq -int$.

Dunque, per la transitività di " \leq ", $''-int \leq '-int$. Perciò, per (r-val), $\Gamma \vdash E : '-int$.

(assign) Si ha la seguente situazione:

$$\text{(assign)} \quad \frac{\Gamma \vdash E : ''-int \text{ var} \quad \Gamma \vdash E' : ''-int \quad -int \leq ''-int}{\Gamma \vdash E := E' : -cmd}$$

Dobbiamo provare che $\Gamma \vdash E := E' : '-cmd$, con $-cmd \leq '-cmd$. Per definizione di " \leq ",

$$-cmd \leq '-cmd \text{ implica} \quad '-int \leq -int$$

Dunque, per transitività $'-int \leq ''-int$, e riapplicando (assign) otteniamo $\Gamma \vdash E := E' : '-cmd$.

(compose), (if), (while) si dimostrano in maniera simile.

(let) Dalla regola (let) segue:

$$(let) \frac{\Gamma \vdash E : \text{"-int} \quad \Gamma[x : \text{"-int var}] \vdash C : \text{-cmd}}{\Gamma \vdash let x := E in C : \text{-cmd}}$$

Da $\text{-cmd} \leq \text{'-cmd}$ e $\Gamma[x : \text{"-int var}] \vdash C : \text{-cmd}$ segue, per ipotesi induttiva, su $\Gamma[x : \text{"-int var}] \vdash C : \text{'-cmd}$. Dunque è possibile concludere $\Gamma \vdash let x := E in C : \text{'-cmd}$ riapplicando la regola (let).

Lemma 2 (Sicurezza). Se $\Gamma \vdash E : \text{-int}$ e $x \notin \text{fv}(E)$ allora $\Gamma(x) = \text{'-int var}$ e $\Gamma' \leq \Gamma$.

Prova: Per induzione sulla struttura sintattica di E.

(variabile) Sia $E = x$, l'albero di prova per dimostrare $\Gamma \vdash x : \text{-int}$ è:

$$\begin{array}{c}
 \Gamma(x) = \text{'-int var} \\
 \text{(ide) } \frac{\text{-----}}{\Gamma \vdash x : \text{'-int var}} \qquad \text{'-int} \leq \text{-int} \\
 \text{(r-val) } \frac{\text{-----}}{\Gamma \vdash x : \text{-int}}
 \end{array}$$

Dalle premesse risulta $\Gamma(x) = \text{'-int var}$ e $\text{'-int} \leq \text{-int}$. Dunque la tesi.

(espressione) Sia $E = E' \# E''$, allora $\Gamma \vdash E : \text{-int}$ segue da (expr#) con premesse $\Gamma \vdash E' : \text{-int}$ e $\Gamma \vdash E'' : \text{-int}$. Per ipotesi induttiva, ogni variabile che occorre in E' ed E'' deve soddisfare la tesi del lemma. Da cui la tesi per $E' \# E''$.

Lemma 3 (Integrità). *Se $\Gamma \vdash C : \text{-cmd}$, allora ogni variabile x libera assegnata in C deve essere tale che $\Gamma(x) = \text{'-int var}$ e $\text{-int} \leq \text{'-int}$.*

Prova: Per induzione sulla struttura sintattica di C . L'unico caso di base è l'assegnamento, gli altri sono casi induttivi.

(assegnamento) Sia $\Gamma \vdash x := E : -cmd$. L'albero di prova di questo judgment è:

$$\begin{array}{c}
 \Gamma(x) = \text{'-int var} \\
 \text{(ide)} \frac{}{\Gamma \vdash x : \text{'-int var}} \\
 \text{(assign)} \frac{\Gamma \vdash x : \text{'-int var} \quad \Gamma \vdash E : \text{'-int} \quad \text{-int} \leq \text{'-int}}{\Gamma \vdash x := E : -cmd}
 \end{array}$$

da cui la tesi del lemma.

(comp. sequenziale) Segue in modo immediato dalle ipotesi induttive sui componenti.

(condizionale) Simile all' (iterazione) di sotto.

(iterazione) Sia $\Gamma \vdash \text{while } E \text{ do } C : -cmd$. La dimostrazione di questo judgment è:

$$\text{(assign)} \frac{\Gamma \vdash E : \text{'-int} \quad \Gamma \vdash C : \text{'-cmd} \quad \text{-int} \leq \text{'-int}}{\Gamma \vdash \text{while } E \text{ do } C : \text{-cmd}}$$

Per ipotesi induttiva, ogni x assegnata in C deve avere $\Gamma(x) = \text{'-int var}$ con $\text{'-int} \leq \text{'-int}$. Da cui il lemma, per transitività di " \leq ".

(blocco) Sia $\Gamma \vdash \text{let } x := E \text{ in } C : \text{-cmd}$. L'ultima regola dell'albero di prova di questo judgment deve essere una istanza di (let):

$$\text{(let)} \frac{\Gamma \vdash E : \text{'-int} \quad \Gamma[x : \text{'-int var}] \vdash C : \text{-cmd}}{\Gamma \vdash \text{let } x := E \text{ in } C : \text{-cmd}}$$

Per ipotesi induttiva applicata a $\Gamma[x : \text{'-int var}] \vdash C : \text{-cmd}$, segue che ogni assegnamento a variabili libere in C soddisfa i requisiti del lemma (si osservi che se x è assegnata in C , allora $\text{-int} \leq \text{'-int}$).

Ciò ci consente di concludere perchè l'assegnamento $x := E$ non deve essere controllato, visto che riguarda una variabile legata.

Esempio. Sia $a : H\text{-int}$ e $b : L\text{-int}$. Allora

$$[y : L\text{-int var}] \vdash \text{let } x := a \text{ in } y := b : L\text{-cmd}$$

sebbene " $x := a$ " abbia tipo " $H\text{-cmd}$ ". Invece

$$[y : L\text{-int var}] \vdash \text{let } x := a \text{ in } y := b ; x := a : L\text{-cmd}$$

poichè " $y := b ; x := a$ " ha tipo " $L\text{-cmd}$ ".

Osservazione. Nel lavoro di Volpano et al. il sistema dei tipi ed il linguaggio sono differenti perchè introducono le locazioni. Queste sono "variabili libere" (non possono essere introdotte da un blocco). In quel contesto, il lemma 3 fa riferimento a locazioni.

Il Lemma di Sostituzione

È un lemma cruciale in tutti i sistemi di verifica statica perchè riguarda l'identità delle variabili.

I nomi delle variabili nei programmi sono simbolici, e sono modificati durante la compilazione/esecuzione.

Tali modifiche sono essenziali nella valutazione dei blocchi, per evitare che il nome della variabile dichiarata collidi con quello di una variabile globale.

Il Lemma di Sostituzione garantisce che tali ridenominazioni non alterano la proprietà di un programma di essere "ben-tipato".

Notazione: $P\{x/y\}$ indica la sostituzione con x di ogni occorrenza libera di y in P

Lemma (Substitution Lemma). Se $\Gamma \vdash x : \text{'-int var}$ e $\Gamma[y : \text{'-int var}] \vdash P : \dots$, allora $\Gamma \vdash P \{x/y\} : \dots$.

Prova: Per induzione sulla struttura sintattica di P . I due casi di base sono le variabili ed i numeri interi, gli altri sono casi induttivi.

Caso di base. ($P = z$) Ci sono due judgment possibili

(1) $\Gamma[y : \text{'-int var}] \vdash z : \text{-int var}$

(2) $\Gamma[y : \text{'-int var}] \vdash z : \text{-int}$.

(1) Siano $\Gamma \vdash x : \text{'-int var}$ e $\Gamma[y : \text{'-int var}] \vdash z : \text{-int var}$. Ci sono due sottocasi, a seconda che $y = z$ oppure $y \neq z$.

a. Nel primo, deve essere $y = z$. Dunque $\Gamma \vdash y \{x/y\} : \text{-int var}$ coincide con l'ipotesi $\Gamma \vdash x : \text{-int var}$.

b. Quando $y \neq z$, per dimostrare $\Gamma[y : \text{'-int var}] \vdash z : \text{-int var}$ utilizziamo la seguente prova

$$(ide) \frac{\Gamma[y : \text{'-int var}](z) = \text{-int var}}{\Gamma[y : \text{'-int var}] \vdash z : \text{-int var}}$$

Per definizione di ambiente, poichè $y \quad z$, $\Gamma[y : \text{'-int var}](z) = \Gamma(z)$. Da cui la tesi.

(2) Siano $\Gamma \vdash x : \text{'-int var}$ e $\Gamma[y : \text{'-int var}] \vdash z : \text{-int}$. La dimostrazione di $\Gamma[y : \text{'-int var}] \vdash z : \text{-int}$ è:

$$(r-val) \frac{\Gamma[y : \text{'-int var}] \vdash z : \text{"-int var} \quad \text{"-int} \leq \text{-int}}{\Gamma[y : \text{'-int var}] \vdash z : \text{-int}}$$

L'ipotesi $\Gamma[y : \text{'-int var}] \vdash z : \text{"-int var}$, assieme a $\Gamma \vdash x : \text{'-int var}$, consente di ridursi al caso (1).

(P n) Immediata conseguenza della regola (int).

Caso induttivo. (P $E \# E'$) Siano $\Gamma \vdash x : \text{'-int var}$ e $\Gamma[y : \text{'-int var}] \vdash E \# E' : \text{-int}$. Per la regola (expr#), devono valere $\Gamma[y : \text{'-int var}] \vdash E : \text{-int}$ e $\Gamma[y : \text{'-int var}] \vdash E' : \text{-int}$. La tesi segue per ipotesi induttive quando si considerano questi ultimi due judgment con $\Gamma \vdash x : \text{'-int var}$.

(P $E := E'$) Siano $\Gamma \vdash x : \text{'-int var}$ e $\Gamma[y : \text{'-int var}] \vdash E := E' : \text{-cmd}$. La dimostrazione di quest'ultimo judgment è:

$$\begin{array}{c} \Gamma[y : \text{'-int var}] \vdash E : \text{"-int var} \quad \Gamma[y : \text{'-int var}] \vdash E' : \text{"-int} \\ \text{-int} \leq \text{"-int} \\ \text{(assign)} \quad \frac{\text{-----}}{\Gamma[y : \text{'-int var}] \vdash E := E' : \text{-cmd}} \end{array}$$

Poichè $(E := E') \{x/y\}$ è per definizione uguale a $E\{x/y\} := E'\{x/y\}$, il lemma segue per ipotesi induttiva.

(P $C ; C', \text{if } E \text{ then } C \text{ else } C', \text{while } E \text{ do } C$) Simile a sopra.

(P **let z := E in C**) Siano $\Gamma \vdash x : \text{'-int var}$ e $\Gamma[y : \text{'-int var}] \vdash \text{let } z := E \text{ in } C : \text{-cmd}$. Per dimostrare quest'ultimo judgment abbiamo:

$$\begin{array}{c}
 \text{(a)} \quad \Gamma[y : \text{'-int var}] \vdash E : \text{"-int} \quad \Gamma[y : \text{'-int var}][z : \text{"-int var}] \vdash C : \text{-cmd} \quad \text{(b)} \\
 \text{(letvar)} \quad \frac{\quad}{\Gamma[y : \text{'-int var}] \vdash \text{let } z := E \text{ in } C : \text{-cmd}}
 \end{array}$$

Ci sono due casi, a seconda che (1) $y = z$ oppure (2) $y \neq z$.

(1) In questo caso, per definizione di sostituzione,

$$(\text{let } z := E \text{ in } C) \{x/z\} = \text{let } z := E\{x/z\} \text{ in } C$$

e, per definizione di ambiente, $\Gamma[y : \text{'-int var}][z : \text{"-int var}] = \Gamma[z : \text{"-int var}]$. Perciò si devono verificare sia $\Gamma \vdash E\{x/z\} : \text{"-int}$ che $\Gamma[z : \text{"-int var}] \vdash C : \text{'-cmd}$. Il secondo judgment segue direttamente da (b), mentre il primo segue per ipotesi induttive da (a) e $\Gamma \vdash x : \text{'-int var}$.

(2) In questo caso, per definizione di sostituzione, ci sono due sottocasi:

- 2.1. x : $(\text{let } z := E \text{ in } C) \{X/Y\} = \text{let } z := E\{X/Y\} \text{ in } C\{X/Y\}$
- 2.2. $x = z$: $(\text{let } z := E \text{ in } C) \{X/Y\} = \text{let } w := E\{X/Y\} \text{ in } C\{W/Z\}\{X/Y\}$
 con w variabile che non appare in E e in C

Nel sottocaso 2.1, si devono provare i due judgment $\Gamma \vdash E\{X/Y\} : \text{"-int}$ e $\Gamma[z : \text{"-int var}] \vdash C\{X/Y\} : \text{'-cmd}$. Si osservi che entrambi seguono per ipotesi induttive usando $\Gamma \vdash x : \text{'-int var}$ ed (a) e (b).

Nel sottocaso 2.2, si devono provare i due judgment $\Gamma \vdash E\{X/Y\} : \text{"-int}$ e $\Gamma[w : \text{"-int var}] \vdash C\{W/Z\}\{X/Y\} : \text{'-cmd}$. Il primo segue per ipotesi induttive usando $\Gamma \vdash x : \text{'-int var}$ ed (a). Per il secondo, applicando il lemma di Weakening a (b) si ottiene:

$$\Gamma[y : \text{'-int var}][z : \text{"-int var}][w : \text{"-int var}] \vdash C : \text{-cmd} \quad (c)$$

Quindi, per ipotesi induttive applicate a (c) ed a

$$\Gamma[y: \text{'-int var}][w: \text{"-int var}] \vdash w: \text{"-int var}$$

Otteniamo

$$\Gamma[y: \text{'-int var}][w: \text{"-int var}] \vdash C\{w/z\}: \text{-cmd}$$

Quindi si ragiona come nel sottocaso 2.1

Problema. le proprietà che abbiamo verificato garantiscono la sicurezza del flusso delle informazioni? *Chi ci assicura che durante l'esecuzione una delle proprietà non sia invalidata?*