

La correttezza al tempo di esecuzione

La tecnica per verificare la “correttezza dinamica” consiste nei seguenti passi:

1. definire l’esecuzione dei programmi (la semantica operativa);
2. dimostrare che se $\Gamma \vdash P$: allora durante l’esecuzione di P non ci sono mai crepe nel flusso (**subject reduction**).

Richiamo. per quanto riguarda 1, le tecniche standard sono due: la semantica naturale (o big step semantics) e la semantica con stati intermedi (o small step semantics).

Entrambe le tecniche richiedono il concetto di configurazione (o stato).

Due tipi (Σ è la *memoria*):

$\Sigma \parallel - P$

Σ

(configurazione iniziale o intermedia)

(configurazione finale)

La memoria Σ è una *funzione parziale a dominio finito dall'insieme delle variabili ai valori* (nel nostro caso solamente interi).

La notazione per accedere alle variabili o modificarne i valori o introdurre nuove variabili è la stessa che per gli ambienti.

Notazione: $\Sigma - x$ è la funzione Σ con il dominio senza la variabile x .

La semantica naturale

Si definiscono le *transizioni* "stato-iniziale stato-finale" in maniera induttiva (come fatto per i judgment dei tipi).

Regole per le espressioni. Formato $\Sigma \Vdash E \quad n$

$$\text{(value)} \quad \Sigma \Vdash n \quad n$$

$$\text{(var)} \quad \frac{\Sigma(x) = n}{\Sigma \Vdash x \quad n}$$

$$\text{(#value)} \quad \frac{\Sigma \Vdash E \quad n \quad \Sigma \Vdash E' \quad n' \quad n \# n' = n''}{\Sigma \Vdash E \# E' \quad n''}$$

{ +, -, =, < }

Commenti

1. non è possibile valutare espressioni le cui variabili non sono definite in memoria
2. la valutazione delle espressioni non modifica la memoria (non ci sono side-effects)
3. l'altezza dell'albero di prova di una transizione $\Sigma \Vdash E$ n è al più uguale al numero di operandi e variabili che occorrono in E. (Il calcolo delle espressioni termina sempre.)

Regole per i comandi. Formato $\Sigma \Vdash C \quad \Sigma'$

$$\text{(update)} \quad \frac{\Sigma \Vdash E \quad n \quad x \quad \text{dom}(\Sigma)}{\Sigma \Vdash x := E \quad \Sigma[x : n]}$$

$$\text{(sequence)} \quad \frac{\Sigma \Vdash C \quad \Sigma' \quad \Sigma' \Vdash C' \quad \Sigma''}{\Sigma \Vdash C ; C' \quad \Sigma''}$$

$$\text{(branch1)} \quad \frac{\Sigma \Vdash E \quad \neq 0 \quad \Sigma \Vdash C \quad \Sigma'}{\Sigma \Vdash \text{if } E \text{ then } C \text{ else } C' \quad \Sigma'}$$

$$\text{(branch0)} \quad \frac{\Sigma \Vdash E \quad 0 \quad \Sigma \Vdash C' \quad \Sigma'}{\Sigma \Vdash \text{if } E \text{ then } C \text{ else } C' \quad \Sigma'}$$

$$\begin{array}{l}
 \text{(loop0)} \quad \frac{\Sigma \Vdash E \quad 0}{\Sigma \Vdash \text{while } E \text{ do } C \quad \Sigma} \\
 \\
 \text{(loop1)} \quad \frac{\Sigma \Vdash E \neq 0 \quad \Sigma \Vdash C \quad \Sigma' \quad \Sigma' \Vdash \text{while } E \text{ do } C \quad \Sigma''}{\Sigma \Vdash \text{while } E \text{ do } C \quad \Sigma''} \\
 \\
 \text{(binder)} \quad \frac{\Sigma \Vdash E \quad n \quad y \in \text{dom}(\Sigma) \quad \Sigma[y : n] \Vdash C\{y/x\} \quad \Sigma'}{\Sigma \Vdash \text{let } x := E \text{ in } C \quad \Sigma' - y}
 \end{array}$$

notazione: Sia $\Gamma \vdash \Sigma$ il judgment definito dalla regola seguente:

$$\frac{\text{per ogni } x \in \text{dom}(\Sigma) : \Gamma \vdash x : \text{var} \text{ e } \Gamma \vdash \Sigma(x) :}{\Gamma \vdash \Sigma}$$

osservazione. Se $\Gamma \vdash \Sigma$ allora $\text{dom}(\Sigma) \subseteq \text{dom}(\Gamma)$.

Un'altra importante proprietà è la seguente:

Lemma di Invarianza. Se $\Gamma \vdash \Sigma$, $\Gamma \vdash C : \text{-cmd}$ e $\Sigma \Vdash C \text{ e } \Sigma'$ allora $\Gamma \vdash \Sigma'$.

Prova: Per induzione sull'altezza dell'albero di prova di $\Sigma \Vdash C \text{ e } \Sigma'$. La dimostrazione è lasciata per esercizio. ■

Il teorema di subject reduction

Vogliamo dimostrare che

1. se un programma è ben tipato
2. una variabile ha tipo $-int\ var$ nell'ambiente

allora non è possibile che il valore finale di questa variabile sia in qualche modo influenzato da variabili che sono di livello di sicurezza maggiore di $-int$.

Teorema (subject reduction). *Siano Γ un ambiente e Σ e Σ' due memorie. Se*

- (a) $\Gamma \vdash \Sigma, \Gamma \vdash \Sigma'$ e $\text{dom}(\Gamma) = \text{dom}(\Sigma) = \text{dom}(\Sigma')$;
- (b) $\Sigma(\mathbf{x}) = \Sigma'(\mathbf{x})$, per ogni \mathbf{x} tale che $\Gamma \vdash \mathbf{x} : -int$;
- (c) $\Gamma \vdash P : \tau, \Sigma \Vdash P \quad \Sigma'', \Sigma' \Vdash P \quad \Sigma''',$

avremo che $\Sigma''(\mathbf{x}) = \Sigma'''(\mathbf{x})$, per ogni \mathbf{x} tale che $\Gamma \vdash \mathbf{x} : \text{-int}$.

osservazione:

1. Per la regola (r-val), $\Gamma \vdash \mathbf{x} : \text{-int}$ significa che $\Gamma(\mathbf{x}) = \text{'-int var}$, con $\text{'-int} \leq \text{-int}$.
2. Il teorema si riferisce solamente alle variabili in Γ , non a quelle che potranno essere introdotte durante l'esecuzione.

Prova: Per induzione sulla struttura dell'albero di prova di $\Sigma \Vdash P$. Mostriamo solamente i casi quando l'ultima regola è (update), (loop0), (loop1) oppure (binder). Gli altri sono trattati in modo simile.

(update) In questo caso $P \quad \mathbf{x} := \mathbf{E}$. Gli alberi di prova di $\Sigma \Vdash \mathbf{x} := \mathbf{E}$
 Σ'' e $\Sigma' \Vdash \mathbf{x} := \mathbf{E}$ Σ''' devono terminare nel modo seguente:

$$\begin{array}{c}
 \Sigma \Vdash E \quad n \quad x \quad \text{dom}(\Sigma) \\
 \text{(update)} \text{-----} \\
 \Sigma \Vdash \mathbf{x} := \mathbf{E} \quad \Sigma[\mathbf{x} : n]
 \end{array}
 \qquad
 \begin{array}{c}
 \Sigma' \Vdash E \quad n' \quad x \quad \text{dom}(\Sigma') \\
 \text{(update)} \text{-----} \\
 \Sigma' \Vdash \mathbf{x} := \mathbf{E} \quad \Sigma'[\mathbf{x} : n']
 \end{array}$$

(perciò $\Sigma'' = \Sigma[\mathbf{x} : n]$ e $\Sigma''' = \Sigma'[\mathbf{x} : n']$). Poichè $\mathbf{x} := \mathbf{E}$ è ben tipato allora deve anche valere:

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

Ci sono due casi, a seconda che $\text{'-int} \leq \text{-int}$ oppure $\text{'-int} \not\leq \text{-int}$

1. $\text{'-int} \leq \text{-int}$. Allora, per il Lemma 2 (Sicurezza) applicato a $\Gamma \vdash \mathbf{E} : \text{'-int}$, ogni variabile y in \mathbf{E} deve avere tipo meno sicuro, cioè

$y : \text{'''-int var}$ e $\text{'''-int} \leq \text{''-int}$, per qualche ''' . Per la transitività, $\text{'''-int} \leq \text{'-int}$, e, per ipotesi $\Sigma(y)=\Sigma'(y)$ su tutte queste variabili. Quindi i valori n ed n' devono essere uguali.

2. $\text{'-int} \not\leq \text{-int}$. Per il judgment $\Gamma \vdash \mathbf{x} : \text{'-int var}$ e la regola (ide) deve essere $\Gamma(\mathbf{x}) = \text{'-int var}$. Quindi, il teorema non si applica a questo caso.

(loop0) e (loop1) In questo caso $P \text{ while } E \text{ do } C$. Poichè P è ben tipato, l'albero di prova deve terminare nel modo seguente:

$$\begin{array}{c}
 \Gamma \vdash E : \text{'-int} \quad \Gamma \vdash C : \text{'-cmd} \quad \text{''} \leq \text{'} \\
 \hline
 \text{(while)} \quad \Gamma \vdash \text{while } E \text{ do } C : \text{''-cmd}
 \end{array}$$

Abbiamo ancora due casi:

1. $'-int \leq -int$. Per il Lemma 2 (Sicurezza), tutte le variabili che occorrono in \mathbb{E} devono avere tipo di livello inferiore a " (e quindi a '). Per lo stesso discorso che si è fatto in (update).1, l'espressione \mathbb{E} sarà valutata allo stesso modo sia in Σ che in Σ' . Perciò avremo

$$\text{(loop0)} \frac{\Sigma \Vdash E \quad 0}{\Sigma \Vdash \text{while } E \text{ do } C} \Sigma$$

$$\text{(loop0)} \frac{\Sigma' \Vdash E \quad 0}{\Sigma' \Vdash \text{while } E \text{ do } C} \Sigma'$$

oppure

$$\text{(loop1)} \frac{\begin{array}{c} \Sigma \Vdash E \quad 1 \quad \Sigma \Vdash C \quad \Sigma_1 \\ \Sigma_1 \Vdash \text{while } E \text{ do } C \quad \Sigma'' \end{array}}{\Sigma \Vdash \text{while } E \text{ do } C} \Sigma''$$

$$\text{(loop1)} \frac{\begin{array}{c} \Sigma' \Vdash E \quad 1 \quad \Sigma' \Vdash C \quad \Sigma_2 \\ \Sigma_2 \Vdash \text{while } E \text{ do } C \quad \Sigma''' \end{array}}{\Sigma' \Vdash \text{while } E \text{ do } C} \Sigma'''$$

Il primo caso ovviamente soddisfa il teorema perchè le memorie non sono modificate.

Nel secondo, per ipotesi induttive, Σ_1 e Σ_2 sono tali che $\Sigma_1(x) = \Sigma_2(x)$, per ogni x tale che $\Gamma \vdash x : -int$.

Inoltre, per il Lemma di Invarianza, $\Gamma \vdash \Sigma_1$ e $\Gamma \vdash \Sigma_2$, e per il Lemma di Stabilità, $\text{dom}(\Gamma) = \text{dom}(\Sigma_1) = \text{dom}(\Sigma_2)$.

Dunque, applicando l'ipotesi induttiva alle premesse

- $\Sigma_1 \Vdash \text{while } E \text{ do } C \quad \Sigma''$
- $\Sigma_2 \Vdash \text{while } E \text{ do } C \quad \Sigma'''$

segue la conclusione.

2. $'-int \not\leq -int$. Per il Lemma di Integrità,

$$\Gamma(\mathbf{x}) = \text{'''-int var} \quad e \quad \text{'''-int} \quad \text{'-int}$$

per ogni x assegnata in C .

Perciò ogni assegnamento di `while E do C` deve riguardare variabili x tali che $\Gamma(x) = \text{"-int var e "-int '-int}$.

Da qui la tesi del teorema.

(binder) In questo caso $P \text{ let } x := E \text{ in } C$. Poichè P è ben tipato, l'albero di prova deve terminare con la seguente regola:

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

Invece gli alberi di prova di $\Sigma \Vdash P \quad \Sigma''$ e $\Sigma' \Vdash P \quad \Sigma'''$ devono terminare così:

$$\begin{array}{c}
\Sigma \Vdash_{\mathbf{E}} \quad n \quad \mathbf{y} \quad \text{dom}(\Sigma) \qquad \Sigma' \Vdash_{\mathbf{E}} \quad n' \quad \mathbf{y} \quad \text{dom}(\Sigma') \\
\Sigma[\mathbf{y} : n] \Vdash_{\mathbf{C}} \{ \mathbf{y}/\mathbf{x} \} \quad \Sigma'' \qquad \Sigma'[\mathbf{y} : n'] \Vdash_{\mathbf{C}} \{ \mathbf{y}/\mathbf{x} \} \quad \Sigma''' \\
\text{(binder)} \frac{\quad}{\Sigma \Vdash \text{let } \mathbf{x} := \mathbf{E} \text{ in } \mathbf{C} \quad \Sigma'' - \mathbf{y}} \qquad \text{(binder)} \frac{\quad}{\Sigma' \Vdash \text{let } \mathbf{x} := \mathbf{E} \text{ in } \mathbf{C} \quad \Sigma''' - \mathbf{y}}
\end{array}$$

Chiaramente

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

per la regola (ide).

Inoltre, poichè $\mathbf{y} \in \text{dom}(\Sigma)$ e, per ipotesi, $\text{dom}(\Sigma) = \text{dom}(\Gamma)$, segue che $\mathbf{y} \in \text{dom}(\Gamma)$.

A questo punto, per il **Lemma di Weakening** e il judgment $\Gamma[\mathbf{x} : \text{'-int var}] \vdash_{\mathbf{C}} \text{'-cmd}$, possiamo derivare

$$\Gamma[\mathbf{y} : \text{'-int var}][\mathbf{x} : \text{'-int var}] \vdash_{\mathbf{C}} \text{'-cmd} \quad (2)$$

(il caso $x=y$ è immediato dal giudizio $\Gamma[x : \text{'-int var}] \vdash c : \text{'-cmd}$
)

Da (1) e (2), per il Lemma di Sostituzione, segue

$$\Gamma[y : \text{'-int var}] \vdash c\{y/x\} : \text{'-cmd}$$

(3)

Inoltre vale $\text{dom}(\Gamma[y : \text{'-int var}]) = \text{dom}(\Sigma[y : n]) = \text{dom}(\Sigma'[y : n'])$.

Per poter applicare l'ipotesi induttiva a (3), dobbiamo verificare che

$$\Sigma[y : n](z) = \Sigma'[y : n'](z)$$

per ogni z tale che $\Gamma[y : \text{'-int var}] \vdash z : \text{'-int}$. Due casi:

- Se $z = y$ allora l'uguaglianza di sopra segue per ipotesi.
- Se $z \neq y$, si deve verificare che $n = n'$ se $\text{'-int} \leq \text{'-int}$.

In tal caso, per il Lemma di Sicurezza applicato a $\Gamma \vdash \mathbb{E} : \text{'-int}$, ogni variabile w che occorre in \mathbb{E} deve essere tale che $\Gamma \vdash$

$w : \text{"-int}$ e $\text{"-int} \leq \text{'-int}$. Su queste variabili Σ e Σ' coincidono, dunque $n = n'$.

A questo punto è possibile applicare l'ipotesi induttiva a (3), ottenendo due memorie Σ'' e Σ''' che coincidono sulle variabili w tali che $\Gamma \vdash w : \text{"-int}$ e $\text{"-int} \leq \text{'-int}$.

A fortiori il teorema vale per $\Sigma'' - y$ e $\Sigma''' - y$. ■

Inferenza di Tipi

Il problema adesso è di progettare un algoritmo che sia in grado di verificare se un programma è ben tipato oppure no.

L' algoritmo, detto **TI** prende in input (Γ, P, V) dove

1. Γ è un ambiente Γ che associa in maniera univoca una variabile di tipo ad ogni variabile libera in P ;
2. P è un programma;
3. V è l' insieme delle variabili di tipo,

e ritorna in output $(\tau, \leq, V \rightarrow W)$ dove

1. τ è un tipo;
2. \leq è un insieme di disegualianze tra tipi;
3. W è un insieme di variabili di tipo,

per cui $\Gamma \vdash P$ e le variabili di tipo in $V \cup W$ sono ordinate secondo \leq .

TI procede in questo modo:

- a) secondo la struttura di P , introduce delle variabili di tipo W non ancora usate (non in V) per tipare il programma P ;
- b) la regola utilizzata per tipare P , imporrà dei vincoli di ordinamento sulle variabili di tipo in V e W ;
- c) se (Γ, \leq, W) è il risultato di **TI**, allora il programma P è ben tipato quando le variabili in W sono istanziate con tipi che rispettino l'ordinamento di \leq ;
- d) si ricordi che

$$\begin{array}{l} \leq \quad \text{allora} \quad -int \leq -int \\ \quad \quad \quad \quad \quad \quad -cmd \leq -cmd \end{array}$$

Notazione.

1. Le variabili di tipo saranno rappresentate dalla lettera τ .
2. Dato un tipo τ , la notazione τ_* è definita come segue:

$$\tau_* = \begin{cases} \tau & \text{se } \tau = \textit{-int} \\ \tau \textit{ var} & \text{se } \tau = \textit{-int var} \\ \tau \textit{ cmd} & \text{se } \tau = \textit{-cmd} \end{cases}$$

3. La notazione $\tau \leq \tau'$ significa $\tau \leq \tau'$ e $\tau' \leq \tau$.

TI (Γ, P, V) =
case P of
n : (*-int*, Γ , V { })
x : *if* $x \in \text{dom}(\Gamma)$ *then* ($\Gamma(x)$, Γ , V)
 else **ERRORE**
E#E' : *let* (Γ', \leq', W') = **TI** (Γ, E, V)
 in let (Γ'', \leq'', W'') = **TI** (Γ, E', W')
 in ($\Gamma', \leq' \leq''$ { $_ = _$ }, W'')
E := E' : *let* (Γ', \leq', W') = **TI** (Γ, E', V) *in*
 if $E = x$ *then*
 if $\Gamma(x) = \text{-int var}$ *then*
 (*-cmd*, \leq' { $_ = _'$, \leq }, W' { })
 else **ERRORE**
 else **ERRORE**

c ; c': *let* (, ≤', W') = **TI** (Γ, c, V)
 in let (', ≤'', W'') = **TI** (Γ, c', W')
 in (, ≤' ≤'' { _ = _' }, W'')

if E then C else C':

let (, ≤', W') = **TI** (Γ, E, V)
 in let (', ≤'', W'') = **TI** (Γ, c, W')
 in let ("", ≤''', W''') = **TI** (Γ, c', W'')
 in (-cmd,
 ≤' ≤'' ≤''' { _ = _' = _'', ≤_ },
 W''' { })

while E do C:

let (, ≤, W) = **TI** (Γ, E, V)
 in let (', ≤', W') = **TI** (Γ, c, W)

in (*-cmd*, \leq \leq' { $_ = _'$, $\leq _ =$ }, *W'* { })

let **x:= E** **in** **C**:

let ($_ , \leq , W$) = **TI** (Γ , E , V)

in let ($_ ' , \leq ' , W'$) = **TI** ($\Gamma [x : _ -int\ var] , C , W$)

in ($_ ' , \leq \leq ' , W'$)

esempio:

1. Sia P `while x > 0 do a := a + 1 ; x := x - 1` e siano $\Gamma = [x : -int\ var, a : -int\ var]$ e $V = \{ , \}$. Sia anche c il corpo del `while`. Allora

$$TI(\Gamma, x > 0, V) = (-int, \{ = \}, V \{ \})$$

$$TI(\Gamma, c, V \{ \}) = (-cmd, \{ \leq, \leq, = \}, V \{ , , \})$$

$$\begin{aligned} TI(\Gamma, P, V) &= (-cmd, \{ \leq, \leq, =, =, \leq \}, V \{ , , , \}) \\ &= (-cmd, \{ \leq, \leq \}, V \{ \}) \end{aligned}$$

2. Sia P `let x:= 0 in a:= a + x; x:= x + 1` e siano $\Gamma = [a : -int\ var]$ e $V = \{ \}$. Allora ... (continuare per esercizio)

Ora bisogna dimostrare che l'algoritmo **TI** è corretto.

In particolare vogliamo dimostrare le due proprietà seguenti:

1. (*consistenza*) Se **TI** (Γ, P, V) non fallisce allora il programma P è ben tipato in un opportuno ordinamento parziale di tipi;
2. (*completezza*) Se P è ben tipato in un sistema di tipi (T, \leq) allora questo sistema di tipi si può evincere dall'output di **TI** (Γ, P, V) .

Un concetto importante per le due proprietà di sopra:

Definizione. Siano (V, \leq) un ordinamento parziale su variabili e (T, \leq') un ordinamento parziale su tipi. Allora (T, \leq') è una istanza di (V, \leq) se esiste una funzione $I : V \rightarrow T$ tale che \leq implica $I(\cdot) \leq' I(\cdot)$.

Teorema di Consistenza. Sia **TI** $(\Gamma, P, V) = (\cdot, \leq, W)$ e (T, \leq_T) sia un'istanza di (W, \leq) con funzione I .

Allora $I(\Gamma) \vdash P : I(\cdot)$, dove $I(\Gamma) = \{ \mathbf{x} : I(\cdot) \text{-int var} \mid \Gamma(\mathbf{x}) = \cdot \text{-int var} \}$.

Prova: Per induzione sulla struttura di P. Analizziamo solamente il caso quando $P \text{ let } \mathbf{x} := \mathbf{E} \text{ in } \mathbf{C}$, gli altri sono più semplici (seguono in maniera diretta dall'ipotesi induttiva).

Poichè $\mathbf{TI}(\Gamma, P, V) = (\Gamma, \leq, W)$, allora, per come è definito l'algoritmo devono essere

$$(\Gamma, \leq_E, W_E) = \mathbf{TI}(\Gamma, \mathbf{E}, V) \quad (1)$$

$$(\Gamma', \leq_C, W_C) = \mathbf{TI}(\Gamma[\mathbf{x} : _int var], \mathbf{C}, W_E) \quad (2)$$

con $\leq = \leq_E \leq_C$ e $W = W_C$. Per definizione dell'algoritmo \mathbf{TI} , (W_E, \leq_E) e (W_C, \leq_C) sono ordinamenti parziali contenuti in (W, \leq) .

Dunque $I(W_E, \leq_E)$ è contenuto in $I(W, \leq)$ ed $I(W_C, \leq_C)$ è contenuto in $I(W, \leq)$. Perciò, poichè $I(W, \leq) = (T, \leq_T)$, (T, \leq_T) è una istanza sia di (W_E, \leq_E) che di (W_C, \leq_C) con funzione I.

Quindi è possibile applicare le ipotesi induttive su (1) e (2), ottenendo le regole

$$I(\Gamma) \vdash E: I(\) \qquad I(\Gamma[x : _ -int\ var]) \vdash C: I(\ ')$$

Il teorema segue applicando la regola (let) a queste due regole.

Teorema di Completezza. Sia $I(\Gamma) \vdash P$: con ordinamento parziale di tipi (T, \leq_T) e con I , funzione di istanza da V a T , dove V è un insieme che contiene le variabili di tipo in Γ .

Allora, $TI(\Gamma, P, V) = (\ ', \leq, W)$ e (T, \leq_T) è un'istanza di (W, \leq) con funzione I' che estende I .

Prova: Per induzione sulla struttura di P . Analizziamo solamente il caso quando P `while E do C`, gli altri sono simili.

Assumiamo che $I(\Gamma) \vdash \text{while } \mathbf{E} \text{ do } \mathbf{c} : \text{-cmd}$ e che le variabili di tipo che occorrono in Γ siano contenute in V .

Per la regola (while), esiste un tipo τ , con $\tau \leq \tau'$, tale che $I(\Gamma) \vdash \mathbf{E} : \tau'$ e $I(\Gamma) \vdash \mathbf{c} : \tau\text{-cmd}$ con lo stesso ordinamento parziale di tipi (T, \leq_T) .

Per ipotesi induttiva, $\mathbf{TI}(\Gamma, \mathbf{E}, V) = (\tau_E, \leq_E, W_E)$ con funzione di istanza I' da W_E a T che estende I . Quindi $I'(\tau_E) = \tau$.

Da $I(\Gamma) \vdash \mathbf{c} : \tau\text{-cmd}$ segue $I'(\Gamma) \vdash \mathbf{c} : \tau\text{-cmd}$ poichè I' estende I e $\text{dom}(I)$ contiene le variabili di Γ .

Perciò è possibile riapplicare le ipotesi induttive, ottenendo $\mathbf{TI}(\Gamma, \mathbf{c}, W_E) = (\tau_C, \leq_C, W_C)$, con funzione di istanza I'' da W_C a T che estende I' .

Quindi $I''(\tau_C) = \tau\text{-cmd}$.

Ci rimane da verificare che

$$(\tau\text{-cmd}, \leq_E \leq_C \quad \{\tau_C = \tau_E, \leq_C\}, W_C \quad \{\})$$

con W_C , soddisfa il teorema con funzione di istanza $I''[\ : \]$.
 Innanzitutto $I''[\ : \]$ estende I per transitività. Inoltre, poichè (T, \leq_T) è istanza sia di (W_E, \leq_E) che di (W_C, \leq_C) , segue che esso è anche istanza di $(W_E \cup W_C, \leq_E \cup \leq_C)$.

Dal fatto che $_$ è nuova, che $W_E \cup W_C$ e che \leq' , per la regola (while), possiamo concludere che (T, \leq_T) è istanza di $(W_C \cup \{ _ \}, \leq_E \cup \leq_C \cup \{ _ = _ \})$ secondo $I''[\ : \]$.