Sintassi e Semantica Nucleo di C:

- Espressioni:
  - numeriche
  - booleane
- Comandi:
  - assegnamento
  - condizionale
  - iteratore
  - sequenza di istruzioni
  - blocco
- Dichiarazioni:

Costrutti comuni a tutti i linguaggi imperativi

Esempio

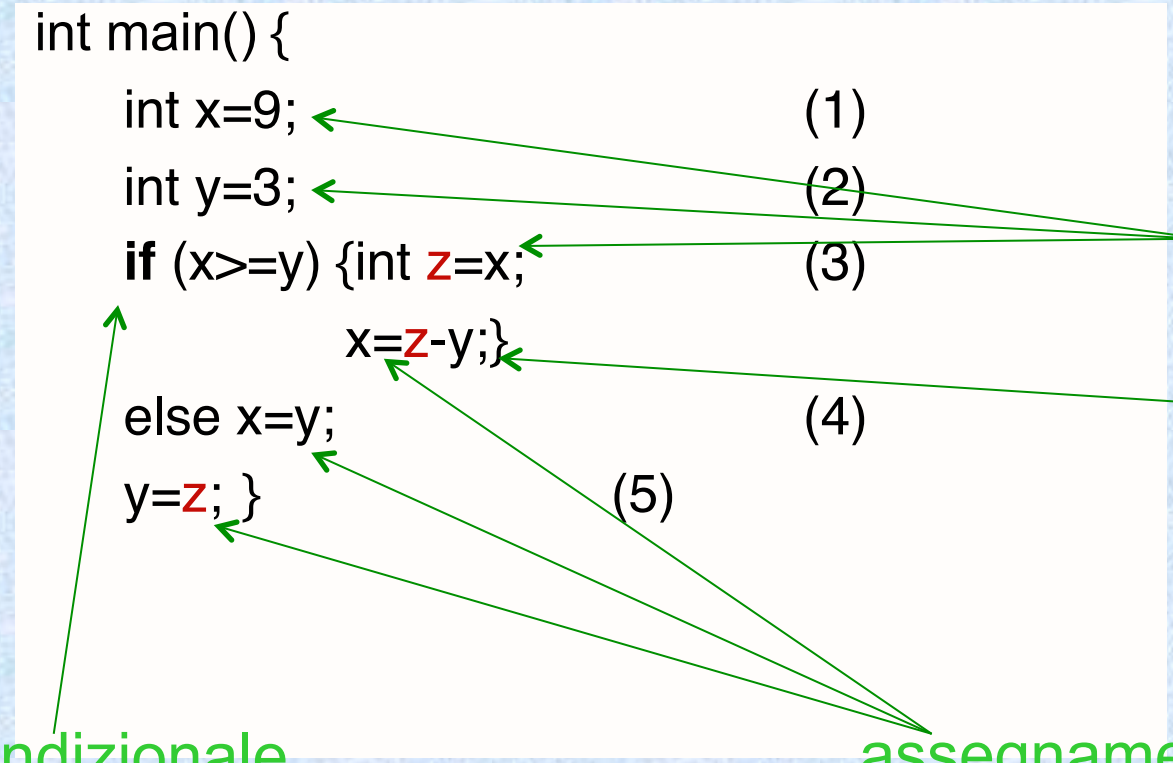
programma nel nucleo di C

```
int main{ int x=20;int y=5; int z=0;
 if (x>y) while (x>y){ x=x-y; z=z+1}
 else if (x<y) while (x<y) {y=y-x; z=z+1;} }
```

programma Caml

```
let f x y = if (x=y) then 0
                else if (x>y) (f (x-y)y)
                else f((y-x) x)
in f(20 5)
```

Un altro esempio



Notare il blocco la dichiarazione e l'uso di z.

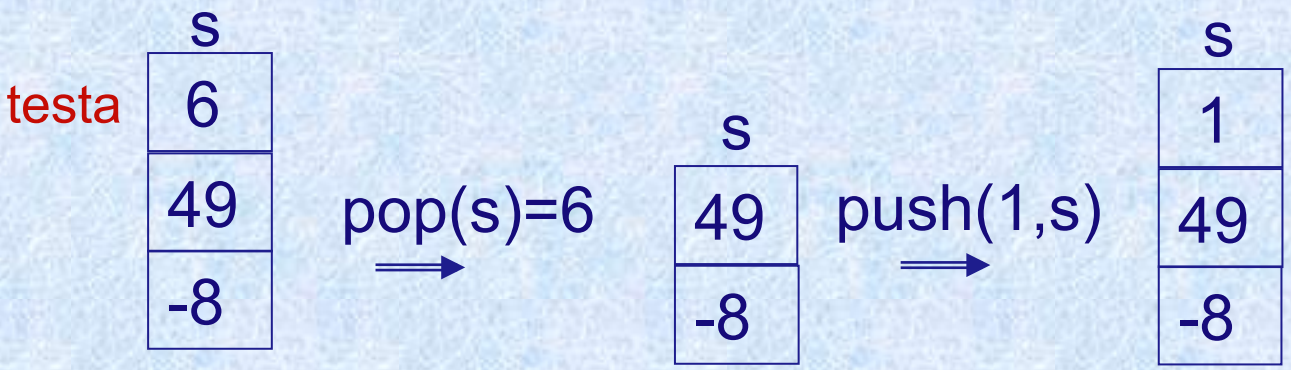
È corretto? Ha senso?

Lo stack (o pila)

(proprio quella dei piatti che avete lavato ieri sera)

- Lo stack è una sequenza di elementi in cui è definito un particolare ordine di manipolazione degli elementi.
- Definiamo
  - lo stack vuoto: è la particolare pila che non contiene elementi (indicata con nil, EmptyStack, Ω, ecc.)
  - la testa dello stack (top dello stack): calcola l'ultimo elemento inserito, non modifica lo stack
  - pop: calcola l'elemento in testa allo stack e modifica lo stack eliminando l'elemento in testa.
  - push: inserisce un elemento in testa allo stack, nel seguito push è indicata con '.'. Modifica lo stack.

Esempi di stack e operazioni



pop(nil)= $\perp$ (indefinito) è una situazione di errore

Lo stato

- Nella semantica che vedremo per il C utilizzeremo i seguenti domini:
  - Ide: gli identificatori validi del C,
  - Val: gli interi
  - Den=Loc ∪ FunctDecl ∪ {Unbound}.
  - Loc: gli indirizzi delle locazioni di memoria (sono naturali)
  - FunctDecl: le dichiarazioni di funzione (le vedremo più avanti

Lo stato

Lo stato ha 2 componenti:

- L'ambiente (nel seguito indicato con σ): che è uno stack di frame. I frame in questo caso rappresentano funzioni da Ide in Den.

- La memoria (nel seguito indicato con μ): che è una funzione dal Loc -> Val. Anche la memoria è un frame.

Operazioni sulle strutture dello stato (Frame)

- Ricerca:
  - $\varphi(i)$ per gli elementi dello stack, ha tipo Φ x Ide $\rightarrow$ Den nell'interprete si chiama look
  - $\mu(\ell)$ per la memoria: ha tipo M x Loc $\rightarrow$ Val nell'interprete si chiama lookM
- Inserzione:
  - free(μ) esiste solo per la memoria e corrisponde all'allocazione di memoria, ha tipo M $\rightarrow$ Loc x M, nell'interprete si chiama free.
  - per gli elementi dello stack $\varphi[d/i]$ ha tipo Φ x Ide x Den$\rightarrow \Phi$
- Modifica:
  - per la memoria $\mu[v/\ell]$ ha tipo M x Loc x Val $\rightarrow$ M, nell'interprete si chiama updateM

Operazioni sugli stack

- Ricerca:
  - σ(i) per gli elementi dello stack, ha tipo Σ x Ide $\rightarrow$ Den nell'interprete si chiama lookUp. Si cerca nel frame top dello stack, se non contiene un legame si va cercare nel resto delo stack, secondo l'ordine di accesso degli elementi usuale per gli stack.
- Inserzione:
  - di un legame (ide,d) ha tipo: Σ x Ide x Den $\rightarrow$ Σ. Nella semantica si esprime come composizione di inserzione di un bind nel frame e push (.), nell'interprete si chiama addBind

Selezione della denotazione associata ad un identificatore nello stack di frame

Formalmente abbiamo la seguente definizione ricorsiva:

$$\sigma(\text{Ide}) = \begin{cases} \varphi(\text{Ide}) & \text{se } \sigma = \varphi.\sigma' \wedge \varphi(\text{Ide}) \neq \perp \\ \sigma'(\text{Ide}) & \text{se } \sigma = \varphi.\sigma' \wedge \varphi(\text{Ide}) = \perp \\ \perp & \text{se } \sigma = \Omega \end{cases}$$

La Grammatica del C

Programma::= *Direttive*

Prototipi

TypeR main() {*StmtList*}

FunctDecList

Direttive ::= *Direttiva Direttive* | ε

Direttiva::= #define *Ide*; | #include <*Ide*.h>;|...

Prototipi::= *Prototipo* ; *Prototipi* | ε

Prototipo:= *Type Ide*(*FormalList*) ;

FunctDecList::= *FunctDecl*; *FunctDecList* | ε

FunctDec::= (int | void) *Ide* (int *Ide*) {*StmtList*}

StmtList::=*Stmt StmtList* | ε

La Grammatica del C - segue

Stmt::=*Decl* | *Com*

Com::=*Ide*=*Exp* *Assegnamento*

| **if** (*Exp*) *Com* **else** *Com* *Condizionale*

| **while** (*Exp*) *Com* *Iteratore*

| {*StmtList*} *Blocco*

| *Ide*(*Exp*) *Invocazione di fun*

| **return** *Exp* *Calcolo dei valore*

Exp::= *Ide* | *Const* | *Exp Op Exp* | *Uop Exp* | *Ide*

| *Ide*(*Exp*) | (*Exp*)...

Decl ::= *Type Ide* [=*Exp*] | *FunctDec*

Type::= int | float |double

TypeR::=*Type* |void

PicoC

- Restrizioni:
  - non ci sono le direttive (anche perchè nella semantica non hanno senso)
  - tipi solo int
  - parametri: funzioni con un e un solo parametro
  - non c'è input/output
  - non ci sono tipi strutturati (array,struct, unioni) ecc).
- Estensioni:
  - le definizioni di funzione sono dichiarazioni e possono comparire in un punto qualunque del programma

PicoC

Programma::=int main() {*StmtList*}

StmtList ::= *Stmt* ; *StmtList* | ε

Stmt::=*Decl* | *Com*

Com::=*Ide*=*Exp* — *Assegnamento*

| **if** (*Exp*) *Com* **else** *Com* — *Condizionale*

| **while** (*Exp*) *Com* — *Iteratore*

| {*StmtList*} — *Blocco*

| *Ide*(*Exp*) — *Invocazione di fun*

| **return** *Exp* — *Calcolo del valore*

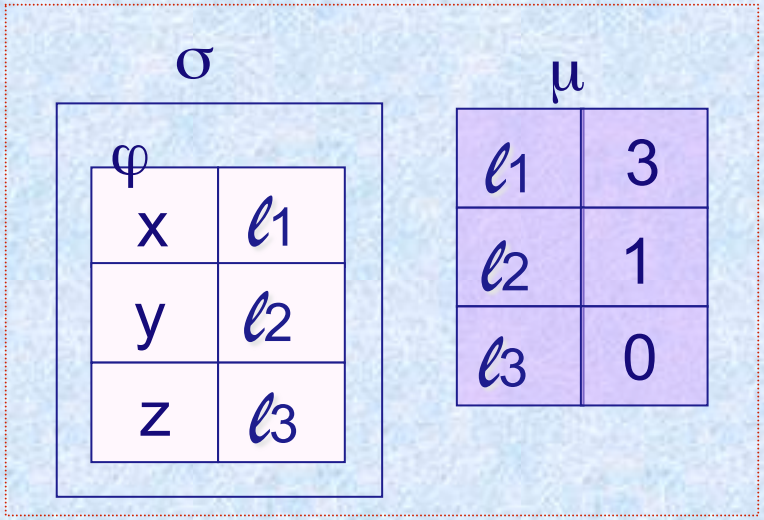
Exp::= *Ide* | *Const* | *Exp Op Exp* | *Uop Exp* | *Ide*

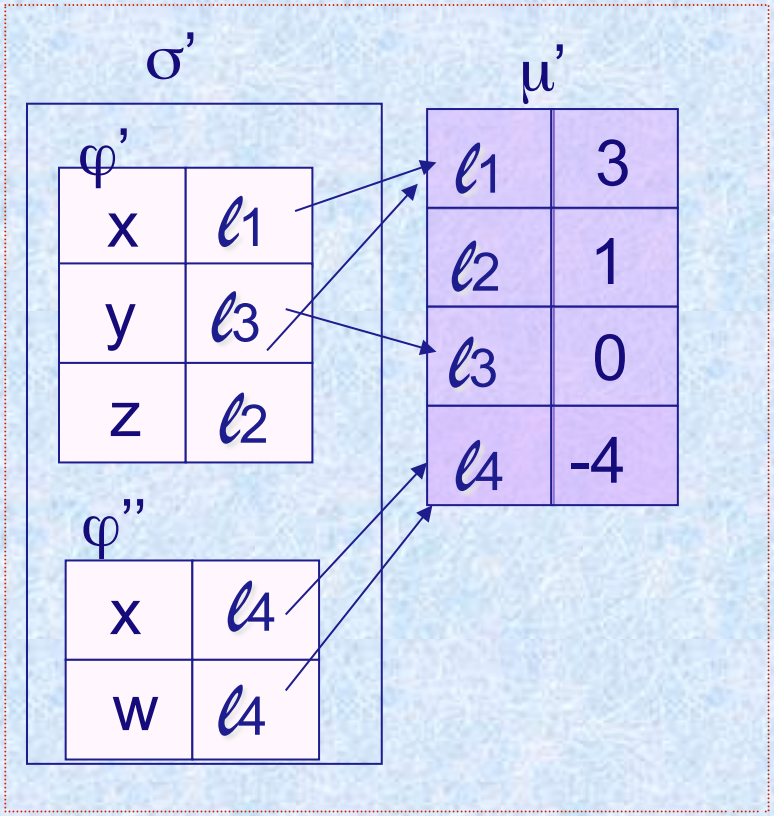
| *Ide*(*Exp*) | (*Exp*)...

Decl ::= int *Ide* [=*Exp*] | *FunctDec*

FunctDec::= (int | void) *Ide* (int *Ide*) {*StmtList*}

Rappresentazione delle variabili nello stato





$\sigma=\{<x, \ell 1>, <y, \ell 2>, <z, \ell 3>\}.\Omega$

$\mu(\sigma(y))=?$

$\mu'(\sigma(y))=?$

$\sigma'=\{\underbrace{<x, \ell 1>, <y, \ell 3>, <z, \ell 2>}_{\varphi'}\}.\{\underbrace{<x, \ell 4>, <w, \ell 4>}_{\varphi''}\}.\Omega$

Sistema di transizioni per le espressioni

$S_{exp} = \{< \Gamma_{exp}, T_{exp}, \rightarrow_{exp} \}$

$\Gamma_{exp} = \{<E,<\sigma,\mu>> \mid E \in Exp, \sigma \in \Sigma, \mu \in M\}$

$\quad \cup \{<v,<\sigma,\mu>> \mid v \in Val, \sigma \in \Sigma, \mu \in M\}$

$T_{exp} = \{<v,<\sigma,\mu>> \mid v \in Val, \sigma \in \Sigma, \mu \in M\}$

$\rightarrow_{exp} = \{(+), \ldots,$ tutte le regole che vedremo $\}$

Semantica: le espressioni

$$(E_{op})\ \frac{\langle E,\langle\sigma,\mu\rangle\rangle \rightarrow_{exp} \langle \underline{v},\langle\sigma,\mu'\rangle\rangle \quad \langle E',\langle\sigma,\mu'\rangle\rangle \rightarrow_{exp} \langle \underline{v}',\langle\sigma,\mu''\rangle\rangle \quad \underline{v}\ [op]\ \underline{v}' = \underline{v}''}{\langle E\ op\ E', \langle\sigma, \mu\rangle\rangle \rightarrow_{exp} \langle \underline{v}'', \langle\sigma, \mu''\rangle\rangle}$$

$$(E_{uop})\ \frac{\langle E, \langle\sigma,\mu\rangle\rangle \rightarrow_{exp} \langle \underline{v}, \langle\sigma, \mu'\rangle\rangle \ [uop]\ \underline{v} = \underline{v}'}{\langle uop\ E, \langle\sigma,\mu\rangle\rangle \rightarrow_{exp} \langle \underline{v}', \langle\sigma, \mu'\rangle\rangle}$$

$$(E_{idev})\ \frac{\mu(\sigma(x)) = \underline{v}}{\langle x, \langle\sigma,\mu\rangle\rangle \rightarrow_{exp} \langle \underline{v}, \langle\sigma,\mu\rangle\rangle}$$

$$(E_{const})\ \frac{v = \eta(const)}{\langle const,\langle\sigma,\mu\rangle\rangle \rightarrow_{exp} \langle \underline{v},\langle\sigma,\mu\rangle\rangle}$$

Sistema di transizioni per i comandi e dichiarazioni

$S_{com} = \{< \Gamma_{com}, T_{com}, \rightarrow_{com} \}$

$\Gamma_{com} = \{<C,<\sigma,\mu>> \mid C \in Stmt, \sigma \in \Sigma, \mu \in M\}$
$\cup \{<\sigma,\mu> \mid \sigma \in \Sigma, \mu \in M \}$

$T_{com} = \{<\sigma,\mu> \mid \sigma \in \Sigma, \mu \in M \}$

$\rightarrow_{com} = \{(=), \ldots,$ tutte le regole che vedremo $\}$

Assegnamento

Sintassi

Com ::= Ide = Exp;

Semantica

$$\frac{\langle E,\langle \sigma,\mu\rangle\rangle \rightarrow_{exp} \langle \underline{v},\langle \sigma', \mu'\rangle\rangle \quad \sigma'(x)=\ell \neq \perp}{\langle x=E;,\langle \sigma,\mu\rangle\rangle \rightarrow_{com} \langle \sigma', \mu'[v/\ell]\rangle} \quad (=)$$

Sequenza di istruzioni: StmtList

Sintassi

StmtList::= Stmt; StmtList | ε

Semantica

$$\frac{\langle Com, \langle \sigma,\mu\rangle\rangle \rightarrow_{com} \langle \sigma'',\mu''\rangle\rangle \quad \langle StmtList,\langle \sigma'', \mu''\rangle\rangle \rightarrow_{com} \langle \sigma',\mu'\rangle\rangle}{\langle Com; StmtList,\langle \sigma,\mu\rangle\rangle \rightarrow_{com} \langle \sigma',\mu'\rangle\rangle} \; (StmtList_{com})$$

$$\frac{\langle Decl,\langle \sigma,\mu\rangle\rangle \rightarrow_{com} \langle \sigma'',\mu''\rangle \quad \langle StmtList,\langle \sigma'',\mu''\rangle\rangle \rightarrow_{com} \langle \sigma'',\mu'\rangle\rangle}{\langle Decl; StmtList,\langle \sigma,\mu\rangle\rangle \rightarrow_{com} \langle \sigma'',\mu'\rangle\rangle} \; (StmtList_{decl})$$

Un condizionale: if

Sintassi

Com::= ... | **if** (BE) Com **else** Com | ...

Semantica

$$\frac{\langle E,\langle\sigma,\mu\rangle\rangle \rightarrow_{exp} \langle tt,\langle\sigma',\mu'\rangle\rangle \quad \langle C_1,\langle\sigma',\mu'\rangle\rangle \rightarrow_{cmd} \langle\sigma'',\mu''\rangle}{\langle \textbf{if}(E)\ C_1\ \textbf{else}\ C_2\ ,\langle\sigma,\mu\rangle\rangle \rightarrow_{com} \langle\sigma'',\mu''\rangle\rangle}\ \text{(if-tt)}$$

$$\frac{\langle E,\langle\sigma\ \mu\rangle\rangle \rightarrow_{exp} \langle ff,\langle\sigma',\mu'\rangle\rangle \quad \langle C_2,\langle\sigma',\mu'\rangle\rangle \rightarrow_{cmd} \langle\sigma'',\mu''\rangle}{\langle \textbf{if}(E)\ C_1\ \textbf{else}\ C_2,\langle\sigma,\mu\rangle\rangle \rightarrow_{com} \langle\sigma'',\mu''}\ \text{(if-ff)}$$

Un iteratore: while

Sintassi

Com::= ... | **while** (BE) Com |...

Semantica

$$\frac{\langle E, \langle \sigma, \mu \rangle \rangle \rightarrow_{exp} \langle ff, \langle \sigma', \mu' \rangle \rangle}{\langle \textbf{while}(E)\ C, \langle \sigma, \mu \rangle \rangle \rightarrow_{com} \langle \sigma', \mu' \rangle} \text{(W-ff)}$$

$$\frac{\langle E, \langle \sigma, \mu \rangle \rangle \rightarrow_{exp} \langle tt, \langle \sigma', \mu' \rangle \rangle \quad \langle C, \langle \sigma', \mu' \rangle \rangle \rightarrow_{com} \langle \sigma'', \mu'' \rangle \quad \langle \textbf{while}(E)\ C, \langle \sigma'', \mu'' \rangle \rangle \rightarrow_{com} \langle \sigma''', \mu''' \rangle}{\langle \textbf{while}(E)\ C, \langle \sigma, \mu \rangle \rangle \rightarrow_{com} \langle \sigma''', \mu''' \rangle} \text{(W-tt)}$$

La semantica: il Blocco

Sintassi

Com::= … | {StmtList} |…

Semantica

$$\text{(Block)}\quad \frac{\langle \text{StmtList}, \langle \omega.\sigma, \mu\rangle\rangle \rightarrow_{com} \langle \varphi.\sigma', \mu'\rangle}{\langle \{\text{StmtList}\}, \langle \sigma, \mu\rangle\rangle \rightarrow_{com} \langle \sigma', \mu'\rangle}$$

$\sigma'=\sigma$? $\sigma'\neq\sigma$?

Semantica delle dichiarazioni

Sintassi

Decl::= Type Ide ; | Type Ide = Exp;

Semantica

(var1)
$$\frac{\text{free}(\mu) = \langle \ell, \mu' \rangle \qquad \sigma = \varphi.\sigma'}{\langle T\ x;, \langle \sigma, \mu \rangle \rangle \rightarrow_{dcl} \langle \varphi\,[\ell/x].\sigma', \mu'[\varpi/\ell] \rangle}$$

(var2)
$$\frac{\langle E, \langle \sigma, \mu \rangle \rangle \rightarrow_{exp} \langle \underline{v}, \langle \sigma, \mu' \rangle \rangle \qquad \text{free}(\mu') = \langle \ell, \mu'' \rangle \qquad \sigma = \varphi.\sigma'}{\langle T\ x=E;, \langle \sigma, \mu \rangle \rangle \rightarrow_{dcl} \langle \varphi\,[\ell/x].\sigma', \mu''[v/\ell] \rangle \rangle}$$

Notare l'inserzione del legame nel top dello stack

Semantica del programma

Sintassi

programma ::= int main () {StmtList}

Semantica

$$\langle \text{StmtList}, \langle \omega.\Omega, \mu_0 \rangle \rangle \rightarrow_{com} \langle \sigma, \mu \rangle$$

(prog)

$$\langle \text{int main() } \{\text{StmtList}\} \rangle \rightarrow \langle \sigma, \mu \rangle$$

Linguaggi a blocchi

- C è un linguaggio a blocchi perchè le variabili esistono solo all'interno del blocco in cui sono definite.
- In C, come nella maggior parte dei linguaggi a blocchi, è corretto riutilizzare un identificatore per definire una variabile, con la seguente semantica:
  - Sia y la variabile definita nel blocco più esterno e y la variabile definita nel blocco interno.
- y e y sono due variabili diverse. La y esiste solo nel blocco interno e poi scompare. La y esiste sempre (*cioè nel blocco esterno e quindi in quello interno*) ma nel blocco interno non si riesce a denotare (è coperta dalla y).
- In Java il riutilizzo di un identificatore in un blocco annidato è vietato

Equivalenza di programmi

- I comandi C e C' sono equivalenti se $\forall \langle \sigma, \mu \rangle$ vale una delle seguenti condizioni:
  1. $\langle C, \langle \sigma, \mu \rangle \rangle$ e $\langle C', \langle \sigma, \mu \rangle \rangle$ NON portano ad una configurazione terminale
  2. Se $\langle C, \langle \sigma, \mu \rangle \rangle \rightarrow^* \langle \sigma', \mu' \rangle$ allora
     $\langle C', \langle \sigma, \mu \rangle \rangle \rightarrow^* \langle \sigma'', \mu'' \rangle$ e vale che
     $\{x \mid x \in Ide \wedge \sigma'(x) \neq \perp\} = \{x \mid x \in Ide \wedge \sigma''(x) \neq \perp\} \wedge$
     $\mu'(\sigma'(x)) = \mu''(\sigma''(x))$
     cioè le configurazioni terminali definiscono le stesse variabili a cui sono assegnati gli stessi valori.

Dimostrazione per casi se necessario

Equivalenza debole (condizionale) di programmi

I comandi C e C' sono equivalenti in uno stato $\langle\sigma,\mu\rangle$ che soddisfa una data condizione cioè tale che ...es $\mu(\sigma(x))=v$), come prima:

Se $\langle C,\langle\sigma,\mu\rangle\rangle \rightarrow^* \langle\sigma',\mu'\rangle$ allora
$\langle C', \langle\sigma,\mu\rangle\rangle \rightarrow^* \langle\sigma'',\mu''\rangle$ e vale che
$\{x \mid x\in Ide \wedge \sigma'(x) \neq \perp\} = \{x \mid x\in Ide \wedge \sigma''(x) \neq \perp\} \wedge$
$\mu'(\sigma'(x)) = \mu''(\sigma''(x))$

cioè le configurazioni terminali definiscono le stesse variabili a cui sono assegnati gli stessi valori.

Dimostrazione per casi se necessario anche in questo caso