ESPRESSIONI IMP:

Exp $E \to A \mid B$

$A \to N \mid A \ op \ A$

$B \to true \mid false \mid not\ B \mid B\ or\ B \mid A = A$

Semantica, Sistema di Transizione

Regole per Expr. Aritmetiche:

- **Assiomi** → si Applicano direttamente
- **Da SX a DX per Valutazione di Expr.**

Procedo **STEP BY STEP** Nella Valutazione fino a che NON raggiungo un Valore Numerico

$(2+3) * (5 - (1+4))$ NON ESISTE QUESTA TRANSIZIONE NELLA DEFINIZIONE DELLE REGOLE

$(2+3) * (5 - \underline{5})$

NON È SBAGLIATA, ma Banalmente NON L'Abbiamo Implementata

Con le Regole con un ordine di Valutazione ho un SISTEMA DETERMINISTICO, altrimenti INSERENDO QUESTA TRANSIZIONE nel Sistema sarà NON DETERMINISTICO visto che dalle Stesse Situazioni ci Sono più possibili Scelte anche se Poi mi Portano allo Stesso Risultato

REGOLE EXP. BOOLEANE:

$T = \mathcal{N} \cup B$ → BOOLEANI

T → TERMINALI

$\mathcal{N}$ → NATURALI

$bop \in \{OR, =\}$ NOT

- NOT $t_0 \to t$ dove $T \in B$ (simboli, quindi SINTATTICA)

  NOT $t_0 = t$ (SEMANTICA)

- t_0 OR $t_1 \to t_2$

  $t_2 = t_1$ OR t_0

- $m = n \to t$

  check Sintattico

  se $m = n$ allora è true (t)

  se $m \neq n$ allora è false (t)

  Valutazione Semantica

$$\frac{e_0 \to e_0'}{\text{NOT } e_0 \to \text{NOT } e_0'}$$

$$\frac{e_1 \to e_1'}{n = e_1 \to m = e_1'}$$

$$\frac{e_0 \to e_0'}{e_0 \ bop \ e_1 \to e_0 \ bop \ e_1'}$$

$$\frac{e_1 \to e_1'}{t \text{ OR } e_1 \to t \text{ OR } e_1'}$$

VALUTAZIONE:

$Eval : exp \to eval$

exp → ESPRESSIONI

$eval$ → VALORI ESPRIMIBILI (o COSTANTI)

È una funzione, INPUT elem. Sintattici e OUTPUT Semantica

$\forall e \in Exp \quad Eval(e) = K \quad$ sse $\quad e \xrightarrow{*}_{\varepsilon} K$

$K \in EVAL$

$\xrightarrow{}_{\varepsilon}$ sistema di Transizione delle Exp.

EQUIVALENZA:

Tra EXP. Senza IDENTIFICATORI

$e_0, e_1 \in EXP$, e_0 è EQUIVALENTE a e_1 $\quad e_0 \equiv e_1 \equiv \subseteq Exp \times Exp$

sse $Eval(e_1) = Eval(e_2)$

IDENTIFICATORI:

È una **SEQUENZA DI CARATTERI** che si usa per DENOTARE qualcosa di DIVERSO rispetto a quella sequenza di Caratteri

ESEMPIO:

```
const pi = 3.14;
int x;
void f() {...}
```

TIPO

OGGETTI DENOTATI

NOMI / IDENTIFICATORI

Ci saranno legami con: (BINDING)

- TIPO → Statico perchè NON cambierà nel NOSTRO linguagg.
- VALORE → Dinamico, è Ammesso che Cambi

ID ⇒ Seq. di Caratteri con Metavariabile ID

ESEMPI DI VINCOLI:

- Case Sensitive o Meno ⇒ x oppure X
- Parole Riservate (NON UTILIZZ. COME IDENTIFICATORI) ⇒ IF, ...
- Sulla lunghezza (LINGUAGGI VECCHI) ⇒ fortran ad Esempio
- Caratteri Speciali ⇒ Es PHP ed il $
- .....

Nel Merito Del Significato Dei Legami

Partendo dalla Matematica:

Qui posso Notare dei LEGAMI

- Occorrenze Di Definizione ⇒ Occorrenze del Nome che ne definiscono il Significato.
  Corrisponde alla Creazione del Significato

- Occorrenze Di Uso / Applicate ⇒ Sto usando dei Nomi ai Quali ho Attribuito un Significato. (ci deve essere)

- Occorrenze Libere ⇒ Non Legate, Sono occorrenze di Nomi a qui NON ho Dato Significato

Un Altro Aspetto Importante: $\left(\sum_{i=1}^{n}\left(\sum_{J=1}^{m} a_{ij}\cdots b_{JK}\right)\right)+i+1$

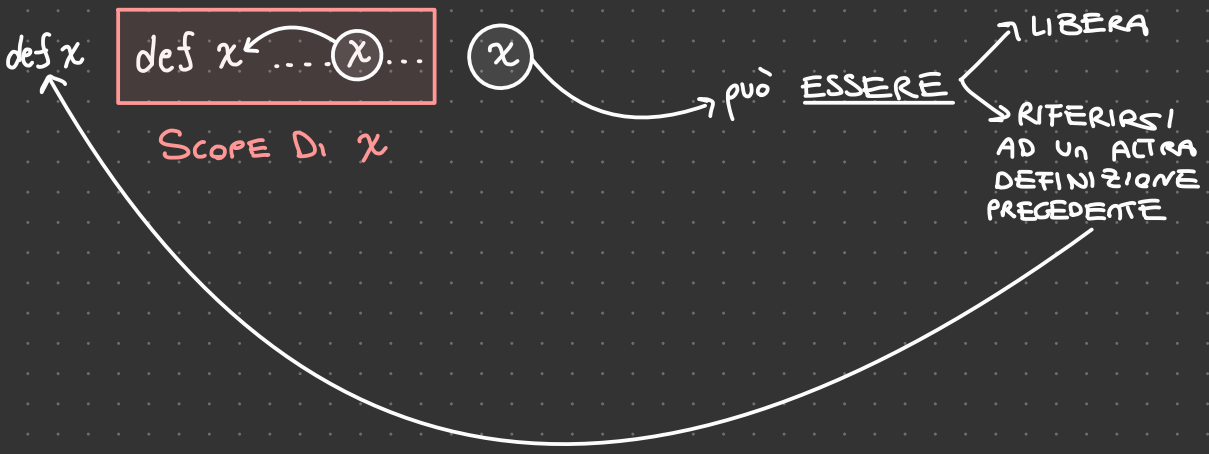
Posso DEFINIRE lo SCOPE di una definizione

→ rapporto di Azione di un occorrenza di definizione di x

↓

Significa che TUTTE le occorrenze di

x Nel Raggio d'azione Sono occorrenze d'uso il cui Significato è definito dalla occorrenza di DEFINIZIONE



BISOGNA MAPPARLI SUI LINGUAGGI DI PROGRAMMAZIONE: (BINDING)

- OCCORRENZA DI DEFINIZIONE ⇒ Quando si Attribuisce un Significato all' Identificatore (DICHIARAZIONI) [Binding]
- OCCORRENZA D'USO ⇒ Quando id è usato per Accedere al suo Significato, quindi se si Trova Nel Raggio d'azione o SCOPE di una sua occorrenza di definizione [applied]
- OCCORRENZA LIBERA ⇒ Non è Nello SCOPE di alcuna occ. di definizione dell' IDENTIFICATORE [free]

Concetto di Legame:

Esistono ≠ TIPI di Binding:

1 `Const pi = 3.14;` → Programma NON Interessa DOVE è Salvata visto che NON verrà MAI Sovrascritta. (rappresenta un VALORE che NON cambia MAI) [nome ↔ Valore]

2 `Var x;` → Deve IGNORARE il Valore di x ma Assegno il nome x all'indirizzo delle celle. Ma Serve un Altro legame per Capire che Valore effettivamente ci sia in x
[Nome ↔ locazione + locazione ↔ Valore]

3 `void f() {...}` → Legame statico, durante il PROGRAMMA Non Viene Modificata la funzione
[nome ↔ Codice di f]

- STATICO ⇒ Non CAMBIA durante L'ESECUZIONE (costante esempio)
- DINAMICO ⇒ Cambia durante L'ESECUZIONE (variabile esempio)

Tempo di Binding: (Quando creo quel legame)

- Tempo di **Compilazione** (early Binding) :- flessibile
- Tempo di **Esecuzione** (Late Binding) : + flessibile

Spesso è una Cosa Mista

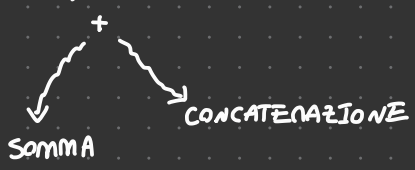
- A Tempo di Progettazione ⇒ Operatori del linguaggio
- A Tempo di Implementazione ⇒ Rappr. dei floating Point
- A Tempo di Compilazione ⇒ I tipi (per linguaggi TIPATI)
- A Tempo di Caricamento (loading) ⇒ id ↔ Celle di Memoria di costanti
- A Tempo di Esecuzione ⇒ id ↔ Celle Memoria per le variabili

Overloading ⇒ Stesso operatore ha **legami diversi**:

- Per la Somma
- Per la Concatenazione

} Viene usato il +

Allora il + è **OVERLOADED**



Identificatore ⇝ Legami con Oggetti Denotati
(BINDING)

AMBIENTE

Insieme dei Legami Tra Identificatori e oggetti DENOTATI

DA CUI DERIVA

occorrenze di definizione

prima la dichiaro

DICHIARAZIONE ⇒ un Meccanismo SINTATTICO ESPLICITO o anche IMPLICITO (non devo dichiararla prima, ma quando la uso viene allocata) per CREARE Legami Tra identificatore e Oggetti Denotati

OCCORRENZE LIBERE non Sono Nel Raggio d'Azione di una dichiaraz.

Termini GROUND ha Sempre Significato ⇒ NON ha Identificatori

Termine CHIUSO se ha Identificatori MA Nessuno è Libero