# Interpreti, compilatori e semantica operazionale



## Linguaggi di programmazione

- Come si comprendono le caratteristiche di un linguaggio di programmazione?
- Molte risposte diverse…
  - manuali, documentazione on-line, esempi, consultazione `stackoverflow`.com, …
- La nostra risposta
  - la comprensione di un linguaggio di programmazione si ottiene dalla **semantica** del linguaggio
- Semantica come guida alla progettazione, all'implementazione e all'uso di un linguaggio di programmazione

# Elementi di semantica operazionale



## Sintassi e semantica

- Un linguaggio di programmazione possiede
  - una **sintassi**, che definisce
    - le "formule ben formate" del linguaggio, cioè i programmi sintatticamente corretti, tipicamente generati da una grammatica
  - una **semantica**, che fornisce
    - un'interpretazione dei "token" in termini di entità (matematiche) note
    - un significato ai programmi sintatticamente corretti
- La **teoria dei linguaggi formali** fornisce formalismi di specifica (grammatiche) e tecniche di analisi (automi) per trattare aspetti sintattici
- Per la semantica esistono diversi approcci
  - **denotazionale, operazionale, assiomatica, …**
- La semantica formale viene di solito definita su una rappresentazione dei programmi in **sintassi astratta**

## Sintassi concreta

- La **sintassi concreta** di un linguaggio di programmazione è definita di solito da una **grammatica libera da contesto** (come visto a **PR1**)
- ***Esempio***: grammatica di semplici **espressioni logiche** (in Backus-Naur Form, BNF)
  - **e ::= v | Not e | (e And e)| (e Or e) | (e Implies e)**
  - **v ::= True | False**
- Notazione comoda per programmatori (operatori infissi, associatività-commutatività di operatori, precedenze)
- Meno comoda per una gestione computazionale (si pensi a problemi di ambiguità)

5

## Sintassi astratta

- L'**albero sintattico** (***abstract syntax tree***) di un'espressione **exp** mostra (risolvendo le ambiguità) come **exp** può essere generata dalla grammatica
- La **sintassi astratta** è una rappresentazione lineare dell'albero sintattico
  - gli operatori sono nodi dell'albero e gli operandi sono rappresentati dai sottoalberi
- Per gli AST abbiamo quindi sia una notazione lineare che una rappresentazione grafica

6

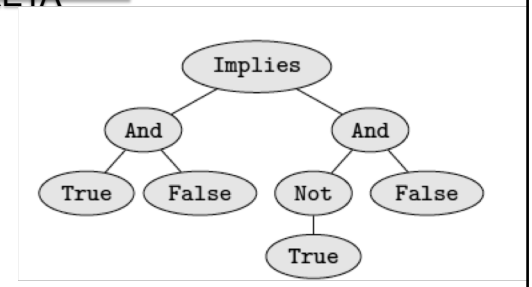
## Dalla sintassi concreta all'AST

**(True And False) Implies ((Not True) And False)**

SINTASSI CONCRETA

**Implies(And(True,False), And(Not(True),False) )**

SINTASSI ASTRATTA




ALBERO SINTATTICO

7

## Semantica dei linguaggi

- Tre metodi principali di analisi semantica
  - ***Semantica operazionale***: descrive il significato di un programma in termini dell'evoluzione (cambiamenti di stato) di una macchina astratta
  - ***Semantica denotazionale***: il significato di un programma è una funzione matematica definita su opportuni domini
  - ***Sematica assiomatica***: descrive il significato di un programma in base alle proprietà che sono soddisfatte prima e dopo la sua esecuzione
- Ogni metodo ha vantaggi e svantaggi rispetto ad aspetti matematici, facilità di uso nelle dimostrazioni, o utilità nel definire un interprete o un compilatore per il linguaggio

8

## E di questo cosa si vede in PR2?

- Metodologie per OOP
  - qualcosa di semantica assiomatica, informalmente
  - clausole Requires & Effect
  - Representation Invariant
- Comprensione dei paradigmi dei linguaggi di programmazione
  - useremo una semantica operazionale

9

## Semantica operazionale

- **Idea:** la semantica operazionale di un linguaggio **L** definisce ***in modo formale, con strumenti matematici,*** una macchina astratta $\mathbf{M_L}$ in grado di eseguire i programmi scritti in **L**
- Definizione: un ***sistema di transizioni*** è costituito da
  - un insieme **Config** di configurazioni (stati)
  - una relazione di transizione $\mathbf{\dashrightarrow \subseteq Config \times Config}$
- Notazione: $\mathbf{c \dashrightarrow d}$ significa che **c** e **d** sono nella relazione $\mathbf{\dashrightarrow}$
- Intuizione: $\mathbf{c \dashrightarrow d}$ lo stato **c** evolve nello stato **d**

10

## Semantica operazionale “small step”

- Nella semantica operazionale “small step” la relazione di transizione descrive un passo del processo di calcolo
- Abbiamo una transizione **e** $-\!\!\rightarrow$ **d** se partendo dall’espressione (o programma) **e** l’esecuzione di un passo di calcolo ci porta nell’espressione **d**
- Una valutazione completa di **e** avrà quindi la forma **e** $-\!\!\rightarrow$ $\mathbf{e_1}$ $-\!\!\rightarrow$ $\mathbf{e_2}$ … $-\!\!\rightarrow$ $\mathbf{e_n}$
  dove $\mathbf{e_n}$ può rappresentare il valore finale di **e**
- Nella semantica *small-step* la valutazione di un programma procede attraverso le configurazioni intermedie che può assumere il programma

11

## Semantica operazionale “big step”

- Nella semantica operazionale “big step” la relazione di transizione descrive la valutazione completa di un programma/espressione
- Scriviamo **e** => **v** se l’esecuzione del programma /espressione **e** produce il valore **v**
- Notazione alternativa (equivalente) utilizzata in molti testi: **e** $\Downarrow$ **v**
- Come vedremo, una valutazione completa di un’espressione è ottenuta componendo le valutazioni complete delle sue sotto-espressioni

12

## Semantica operazionale in PR2

- La visione del corso: utilizzeremo la semantica operazionale “big step” come modello per descrivere i meccanismi di calcolo dei linguaggi di programmazione
- La semantica operazionale “small step” sarebbe utile nel caso di linguaggi concorrenti per descrivere le comunicazioni e proprietà quali la deadlock freedom

13

## Semantica operazionale “big step” di espressioni logiche

$$true \Rightarrow true \qquad false \Rightarrow false$$

**VALORI**

$$\frac{e \Rightarrow v}{not\ \mathrm{e} \Rightarrow \neg v}(not)$$

$$\frac{e1 \Rightarrow v1 \quad e2 \Rightarrow v2}{e1\ \mathrm{and}\ e2 \Rightarrow v1 \wedge v2}(and)$$

**Regole di valutazione analoghe per OR e IMPLIES**
**Usiamo OPERATORI LOGICI sul dominio dei valori con tabelle di verità**

14

## Regole e derivazioni

- Le regole di valutazione possono essere composte per ottenere la valutazione di una espressione più complessa
- Questo fornisce una prova di una derivazione operazionale di calcolo

$$\dfrac{\dfrac{}{\texttt{True} \Rightarrow \texttt{True}} \quad \dfrac{\dfrac{}{\texttt{False} \Rightarrow \texttt{False}} \quad \dfrac{}{\texttt{True} \Rightarrow \texttt{True}}}{\texttt{False And True} \Rightarrow \texttt{False}}}{\texttt{True And (False And True)} \Rightarrow \texttt{False}}$$

15

## Regole di derivazione

- Le regole di valutazione costituiscono un *proof system* (sistema di dimostrazione)

$$\frac{\textit{premessa}_1 \quad \ldots \quad \textit{premessa}_k}{\textit{conclusione}}$$

- Tipicamente le regole sono definite per induzione strutturale sulla sintassi del linguaggio
- Le "formule" che ci interessa dimostrare sono transizioni del tipo **e => v**
- Componiamo le regole in base alla struttura sintattica di **e** ottenendo un *proof tree*

16

## Regole e interprete

- Le regole di valutazione definiscono l'interprete della macchina astratta "formale" definita dalla semantica operazionale
- Quindi le regole descrivono il processo di calcolo
- Nel corso forniremo una codifica OCaml della semantica operazionale
- Di conseguenza otterremo un modello eseguibile dell'interprete del linguaggio

17