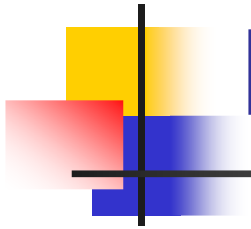
Linguaggi e Laboratorio

Laurea in Ingegneria Telematica – Università Kore di Enna
A.A. 2009-2010

Alessandro Longheu
http://www.diit.unict.it/users/alongheu
alessandro.longheu@diit.unict.it

- lezione 01 -

Introduzione ai linguaggi di programmazione



Definizioni

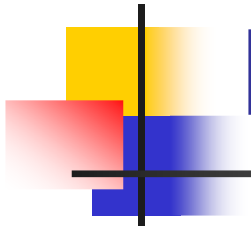
Cos'è un linguaggio di programmazione?

La definizione utilizzata per i linguaggi naturali…

"*Un linguaggio è un insieme di parole e di metodi per combinare le parole usate e comprese da una comunità di persone*"

…è **poco precisa** perché:

- Non evita l'ambiguità tipica dei linguaggi naturali
- Non è adatta a descrivere processi computazionali meccanizzabili
- Non aiuta a definire quali proprietà caratterizzano un linguaggio

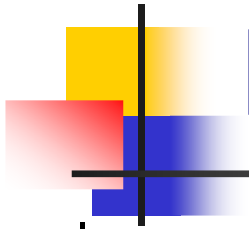


Definizioni

In realtà i linguaggi di programmazione rientrano nella categoria dei **linguaggi formali** (non naturali).

La teoria dei linguaggi fornisce le seguenti definizioni:

- sia dato un insieme finito di **simboli** Σ denominato **alfabeto**; gli elementi dell'alfabeto sono anche detti **simboli terminali**
- una sequenza di simboli $\in \Sigma$ prende il nome di **stringa**
- sia dato un insieme finito di **regole P** (denominate anche **produzioni**) che indicano quali stringhe sono da considerarsi valide in Σ; le regole sono espresse nella forma $\alpha \rightarrow \beta$, con:
  - $\alpha \in V$, insieme dei **simboli non terminali** (perché sono il lato sinistro di una produzione, verranno quindi sostituiti con il lato destro)
  - $\beta \in (\Sigma \cup V)^*$, ossia una sequenza di simboli terminali e non; l'operatore * è noto come **stella di Kleene**; A* indica la ripetizione di 0 o più occorrenze di A
  - le regole sono chiamate anche **produzioni** perché α produce β
  - uno dei simboli non terminali è scelto come iniziale (**assioma**)

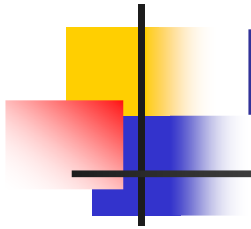


Definizioni

- la coppia **(Σ,P)** viene definita **grammatica**
- data una grammatica **G (Σ,P)**, si definisce **linguaggio L(G)** l'insieme di tutte le possibili stringhe generabili da Σ utilizzando **P**

Esempio di linguaggio L:

- simboli terminali {0,1}
- simboli non terminali
  - S→0A,
  - S→111B
  - A→10,
  - A→1B,
  - B→1
- le stringhe del linguaggio sono 010, 011, 1111; la stringa 000 ad esempio NON appartiene al linguaggio
- un **traduttore** per linguaggi di programmazione applica le produzioni al programma (sequenza di caratteri in ingresso) per convalidarlo (e successivamente tradurlo ed eseguirlo) o per segnalare errori

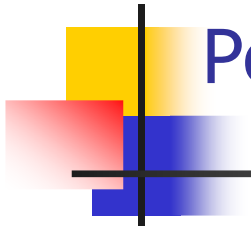


Definizioni

Un **linguaggio di programmazione** è un linguaggio formale utilizzato per implementare gli algoritmi

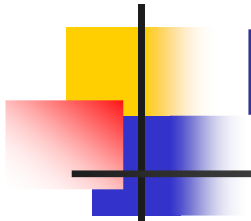
Richiami:

- Un **algoritmo** è un insieme finito di azioni da eseguire per risolvere un problema
- Un **programma** è la descrizione di un algoritmo mediante un opportuno linguaggio (di programmazione)
- Un **processo** è un programma in corso di esecuzione



Perché studiare i linguaggi di programmazione ?

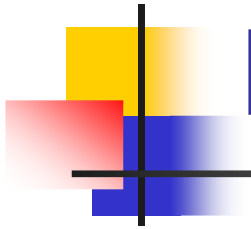
- Accrescere le capacità di esprimere le idee
- miglioramento del background al fine di scegliere il linguaggio più appropriato
- accrescimento della capacità di apprendimento di nuovi linguaggi
- migliore comprensione del significato di implementazione
- accrescere la capacità di progettare nuovi linguaggi



Proprietà di un linguaggio

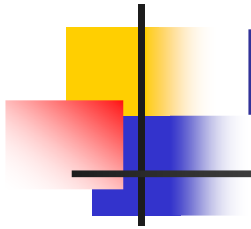
Le **caratteristiche** di un linguaggio di programmazione sono:

- **Potere espressivo**: consente una maggiore immediatezza (meno istruzioni) nell'implementare la soluzione di un problema; il supporto per l'astrazione incrementa il potere espressivo
- **Ortogonalità**: si realizza quando in un linguaggio esistono pochi elementi di base, e combinandoli si ottengono tutti i costrutti.
  - Esempio con i concetti di *tipo* e *funzione*. In un linguaggio ortogonale l'indipendenza fra i due concetti consente di poter avere una funzione può lavorare con parametri di qualsiasi tipo; in un linguaggio non ortogonale (Pascal), esistono invece restrizioni, ad esempio il tipo *record* non può essere restituito da una funzione (non sussiste una reale indipendenza).
  - In un linguaggio ortogonale deve sussitere la *legge della minima sorpresa*: si deve poter dedurre l'effetto di un costrutto conoscendo gli elementi di base del linguaggio.



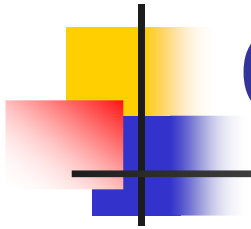
Proprietà di un linguaggio

- **Readability** (leggibilità): dalla lettura di un programma dovrebbe essere facile e naturale seguirne la logica e scoprire l'eventuale presenza di errori. Il requisito di leggibilità di un programma ha assunto un importanza sempre maggiore, poiché il costo della maintenance (fortemente correlato con la leggibilità) è ormai preponderante. Per garantire la leggibilità occorre soddisfare:
  - lettura dall'alto in basso
  - limitazioni all'uso del goto
  - strumenti per la definizione dei tipi di dato
  - naming esteso per gli identificatori
  - promozione nell'uso di parole chiave
- **Writtability** (scrivibilità): è una misura della facilità di utilizzare un linguaggio per lo sviluppo di programmi. Tale proprietà e difficilmente quantizzabile perché, per una sua valutazione, i linguaggi dovrebbero essere paragonati in relazione ad una qualsiasi applicazione.



Proprietà di un linguaggio

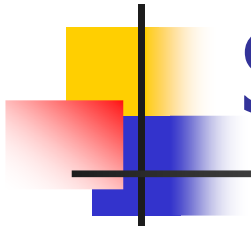
- **Semplicità**: Un linguaggio con un grande numero di componenti elementari è usualmente più difficile da imparare e spesso i programmatori sono portati ad imparare ed usare solo un sottoinsieme del linguaggio.
- **Implementazione**: facilità ed efficienza con cui un programma (scritto in un dato linguaggio) può essere eseguito
- **Standardizzazione/Portabilità (ANSI, ISO, IEEE)**
- **Affidabilità (reliability)**: un linguaggio è affidabile se esso si comporta secondo le specifiche (cioè in modo prevedibile) sotto tutte le condizioni comprese quelle anomale; il linguaggio dovrebbe permettere di riconoscere eventi indesiderati e specificare per ognuno di essi la risposta più adatta. Correlati all'affidabilità sono:
  - Type-checking
  - Rilevazione e correzione degli errori
  - Restricted Aliasing (Aliasing e' la possibilita' di avere due riferimenti distinti per la stessa cella di memoria)



Classificazione

Diversi criteri sono utilizzati per classificare i linguaggi:

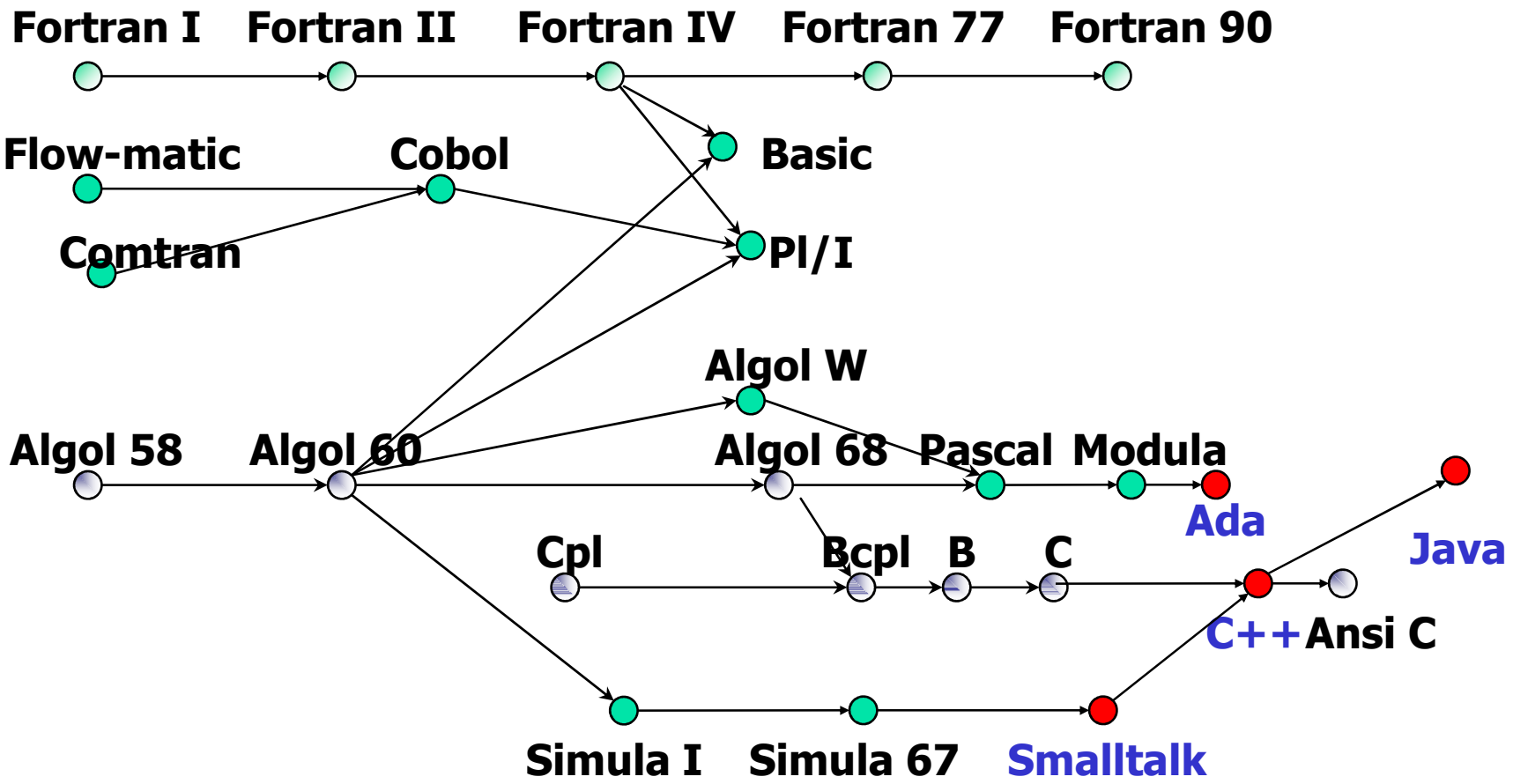
- Periodo storico di definizione
- Program domain
- Struttura della grammatica
- Livello di astrazione
- Paradigma di programmazione

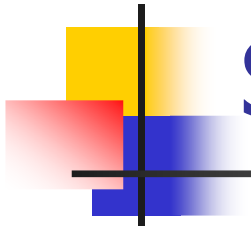


Classificazione
Storia dei linguaggi

Molti linguaggi sono stati sviluppati a partire dagli anni ’50; alcuni di essi sono ancora oggi utilizzati, fondamentalmente a causa dell’**inerzia** che caratterizza il mondo della programmazione: i grandi investimenti fatti da programmatori ed industrie nell’uso di un linguaggio, e la grande quantità di software già presente limitano fortemente l’introduzione di un nuovo linguaggio, anche quando possiede caratteristiche innovative

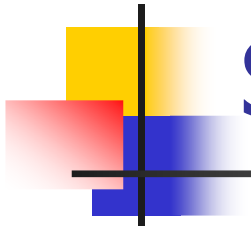
Classificazione
Storia dei linguaggi





Classificazione
Storia dei linguaggi – FORTRAN

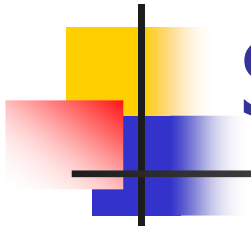
- Il **FORTRAN** (FORmula TRANslating system) è stato ideato nel 1956, si è evoluto in varie versioni, sino al FORTRAN 90
- è stato principalmente progettato per il **calcolo scientifico**, mancano quindi istruzioni specifiche per le stringhe, e strutture dati diverse dall'array (aggiunte in seguito)
- Il FORTRAN ha avuto molto successo, ed è ancora oggi adoperato per diversi fattori:
  - Esistono numerose librerie di calcolo scientifico, accumulatesi negli anni
  - È facile da apprendere, anche per non informatici
  - È stato supportato da IBM per molti anni
  - È stato introdotto in un periodo storico in cui i calcolatori erano utilizzati prevalentemente per applicazioni scientifiche, quindi attirava quasi tutto il parco utenza



Classificazione
Storia dei linguaggi – FORTRAN

Esempio di programma:

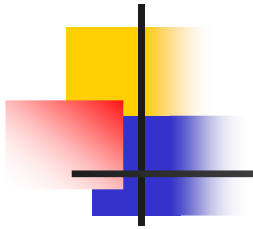
```
  1234567
1       PROGRAM MAIN
2       PARAMETER (MAXSIZ=99)
3       REAL A(MAXSIZ)
4     10 READ(5,100,END=999) K
5    100 FORMAT(I5)
6          IF (K.LE.0.OR.K.GT.MAXSIZ) STOP
7          READ *,(A(I), I=1,K)
8          PRINT *,(A(I), I=1,K)
9          GO TO 10
10   999 PRINT *, “ALL DONE”
11      STOP
12C THIS IS A COMMENT
13      END
```



Classificazione
Storia dei linguaggi – FORTRAN

Commenti:

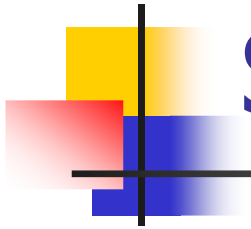
- Il FORTRAN, sviluppatosi quando esistevano ancora le schede perforate, richiede che la colonna 1 sia riservata per la lettera C (la riga è un commento), le colonne dalla 2 alla 5 sono usate per etichette numeriche (label), la 6 è libera (allineamento delle schede perforate), e dalla colonna 7 in poi si scrive il codice
- La riga 2 definisce una costante
- La riga 3 definisce un vettore
- Nella riga 4 si legge il numero di elementi che si vorranno utilizzare per il vettore (da 1 a MAXSIZ); il 5 indica la tastiera, il 100 la label dove trovare l'istruzione FORMAT per formattare l'input da tastiera, la END=999 indica a quale riga saltare in caso di eccezioni (errori sull'input); la variabile K è intera (in FORTRAN lo sono tutte quelle il cui identificatore comincia con I,J,K,L,M)



Classificazione
Storia dei linguaggi – FORTRAN

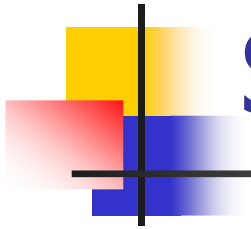
Commenti:

- La riga 5 dice che il formato è un intero e considera 5 caratteri per l'input
- La riga 6 controlla che K sia fra 1 e 99
- La riga 7 legge sequenzialmente K reali e li mette in A(I), mentre la riga 8 stampa il vettore stesso
- La riga 9 mostra che il FORTRAN (almeno le prime versioni) non è strutturato; il goto rimanda indietro, finchè l'utente non immette un valore fuori range per K, verrà letto e stampato un nuovo vettore



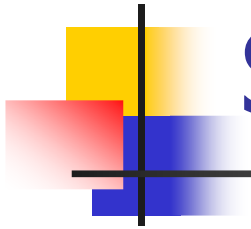
Classificazione
Storia dei linguaggi – ALGOL

- **ALGOL** (ALGorithmic Language) è stato sviluppato da un comitato euro-statunitense nel 1957 con l'obiettivo di produrre un **linguaggio comune**, basato sulla sintassi matematica
- L'ALGOL è il linguaggio che **più di tutti ha influenzato l'informatica**, contribuendo alla nascita di quasi tutti i linguaggi di programmazione (che vengono infatti denominati algol-like), grazie anche al fatto che è stato il primo linguaggio ad essere descritto in forma BNF, ad essere strutturato (divisione in blocchi), e ad introdurre la ricorsione
- Nonostante la sua influenza teorica, non riuscì mai a superare il FORTRAN, anche perché:
  - Venne sviluppato dopo la nascita del FORTRAN, che già era affermato
  - Possiede molte caratteristiche, che lo rendono difficile da imparare
  - È mancato il supporto dell'IBM



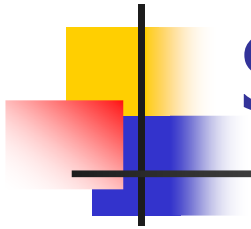
Classificazione
Storia dei linguaggi – COBOL

- **COBOL** (COmmon Business Oriented Language) è stato sviluppato dal Dipartimento della Difesa (DoD) USA nel 1960
- È essenzialmente un linguaggio per **processare dati** e pone particolare enfasi sull'I/O
- I comandi sono molto estesi, per facilitare i non esperti
- Nonostante non sia innovativo, ha introdotto l'idea di separare la descrizione logica dei dati dall'ambiente fisico (file) e dalle procedure che elaborano i dati, garantendo indipendenza fra i tre aspetti; questo si riflette sui programmi, sempre divisi in quattro sezioni:
  - identificazione, per commenti e documentazione;
  - ambiente, che specifica i file coinvolti;
  - dati, che fornisce la descrizione logica dei dati;
  - Procedure
- Il COBOL, anche se ancora oggi usato per le molte procedure e librerie disponibili, non ha saputo adattarsi alle esigenze sorte nel corso del tempo, restando sostanzialmente identico alla prima versione



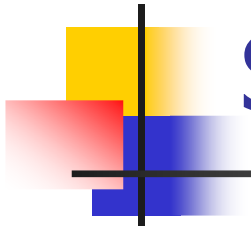
Classificazione
Storia dei linguaggi – PL/I

- Successivamente all'introduzione di FORTRAN e COBOL, l'esigenza di integrarne le potenzialità, aggiungendo anche ulteriori innovazioni, portò alla creazione di diversi linguaggi; fra questi **linguaggi multipurpose**, il Jovial ed il PL/I
- Il **PL/I** venne introdotto dall'IBM insieme al sistema IBM360; e combina molte delle idee di FORTRAN, COBOL ed ALGOL (capacità di calcolo, gestione dell'I/O, strutturazione), aggiungendo anche, fra l'altro, gestione delle eccezioni ed elaborazione di liste
- Il PL/I riusciva a combinare l'efficienza a run-time (tipica del FORTRAN e COBOL) con una buona flessibilità (come nel LISP), a prezzo di un incremento di complessità; in questo senso è uno dei linguaggi più completi



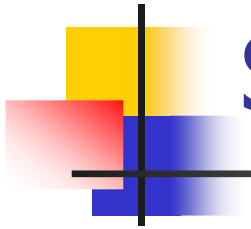
Classificazione
Storia dei linguaggi – Panoramica

- Il **BASIC** (Beginner All-purpose Symbolic Instruction Code) venne sviluppato nella metà degli anni 60 con l'intento di essere estremamente facile da apprendere; Ebbe un discreto successo, anche per la diffusione che riuscì ad avere negli ani 80, in cui ebbe inizio la vendita dei microcomputer (PC)
- Il **LISP**, ideato alla fine degli anni 50, è un linguaggio in cui i tipi di dato, finora tendenzialmente solo array, possono essere molto complessi (liste annidate); viene principalmente utilizzato per il calcolo simbolico e nel campo dell'intelligenza artificiale
- **SIMULA** invece è un linguaggio, nato all'inizio degli anni 60, dedicato alla simulazione dei sistemi discreti, in cui parallelismo ed interazione sono determinanti
- **BCPL** ed **Occam** furono due dei linguaggi utilizzati per la scrittura dei sistemi operativi, in cui era fondamentale la capacità da parte del linguaggio di accedere direttamente alle risorse della macchina (lavorare a basso livello)



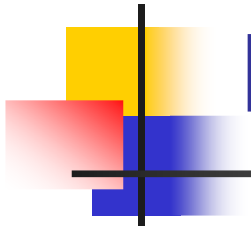
Classificazione
Storia dei linguaggi – Panoramica

- Il **PASCAL**, sviluppato alla fine degli anni 60, era stato concepito per l'insegnamento della programmazione, mantenendo nel contempo l'efficienza del linguaggio
- Il **C** deriva dal BCPL e dal linguaggio B, sviluppati per la scrittura di UNIX; combina peculiarità di alto livello insieme all'efficienza dell'assembler (possiede puntatori, operatori su bit ecc.)
- **ADA**, sviluppato dal DoD USA nella metà degli anni 70, è un linguaggio per sistemi embedded, con alto grado di parallelismo e di variabilità, e la necessità di essere fault tolerant, con un'ampia gestione delle eccezioni
- **Modula-2** è un'estensione del Pascal con istruzioni per la multiprogrammazione e la concorrenza, più elementare comunque di ADA



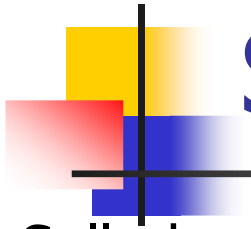
Classificazione
Storia dei linguaggi – Panoramica

- Il **PROLOG** è un linguaggio logico, basato su asserzioni piuttosto che su istruzioni, dalla quali la macchina, tramite un motore inferenziale, determina la soluzione di un dato problema
- **RPG** è una semplificazione del COBOL, ideata per ottenere una facile generazione di report su insiemi di dati (output formattato)
- **SNOBOL** è un linguaggio specializzato per la manipolazione delle stringhe tramite pattern matching



Classificazione
Program domain

- **Applicazioni scientifiche**
  - sono caratterizzate da semplici strutture dati e grandi quantità di calcoli su floating point (FORTRAN, C, C++)
- **Applicazioni gestionali**
  - sono caratterizzate da sofisticate caratteristiche di input/output (COBOL)
- **Intelligenza artificiale**
  - sono applicazioni caratterizzate da processamenrto simbolico e dall'uso delle liste come tipo di dato primitivo (PROLOG, Lisp)
- **Programmazione di sistemi**
  - sono caratterizzate dalla necessità di operare a basso livello, esecuzione efficiente (Assembly, C)
- **Linguaggi special-purpose**
  - non possono essere individuate caratteristiche comuni (Snobol, RPG)



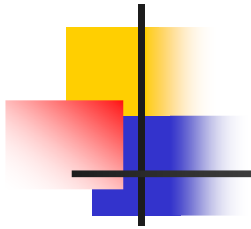
Classificazione
Struttura grammatica

Sulla base della struttura della grammatica (forma delle produzioni), il linguista Noam Chomsky ha fornito una **classificazione dei linguaggi** e delle relative grammatiche che li generano:

- Tipo 0 – **Linguaggi generali** (enumerabili ricorsivamente), riconosciuti dalle *macchine di Turing*
- Tipo 1 – **Linguaggi dipendenti dal contesto**, riconosciuti da *automi linearmente limitati* (macchine di Turing con nastro di lunghezza finita)
- Tipo 2 – **Linguaggi non contestuali**, riconosciuti dagli *automi a pila*
- Tipo 3 – **Linguaggi regolari**, riconosciuti dagli *automi a stati finiti*

I linguaggi di tipo 3 sono i più restrittivi, fino al tipo 0 che non presenta nessuna limitazione

Un linguaggio di tipo N è anche di tipo N-1, il viceversa non è vero



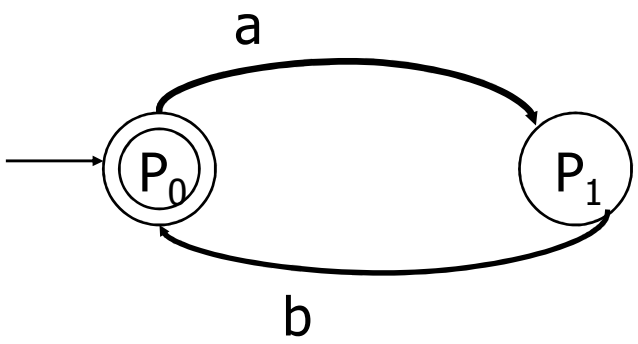
Classificazione
Struttura grammatica
Linguaggi regolari (tipo 3)

- Le grammatiche di tipo 3 sono molto restrittive, e quindi non si possono utilizzare per la sintassi dei linguaggi di programmazione, tuttavia i loro riconoscitori sono molto efficenti.
- Il riconoscitore di un linguaggio di tipo 3 è un **automa a stati finiti** (finite-state automaton, o FSA), definito come una quintupla (S, Σ, T, Start, FS), dove:
  - S è l'insieme degli stati
  - Σ è l'insieme dei simboli ammessi
  - T è l'insieme delle transizioni di stato, ognuna associata ad un simbolo
  - Start è uno degli stati che viene scelto come iniziale
  - FS è un sottoinsieme di S che rappresenta i possibili stati finali
- L'automa prevede che, partendo dallo stato iniziale, viene letto un simbolo di ingresso, ed avviene in corrispondenza una transizione di stato, procedendo fino al raggiungimento di uno stato finale (avvenuto riconoscimento della stringa in ingresso).

Classificazione
Struttura grammatica
Linguaggi regolari (tipo 3)

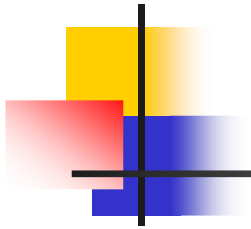
Esempio di FSA per la grammatica ({a,b}, {S → ε, S → abS}) :

- S = { P_0, P_1 }
- Σ = { a, b }
- T = { P_0→ P_1, P_1→ P_0 }
- Start = { P_0 }
- FS = { P_0 }



Per ogni linguaggio di tipo 3, esiste un corrispondente FSA opportunamente ricavabile dal suo alfabeto e dalle sue produzioni

Gli FSA sono definiti **deterministici (DFA)** quando, dato un possibile simbolo di ingresso, esiste una sola transizione uscente da ogni stato, altrimenti è detto **non deterministico (NFA)**; Kleene ha dimostrato che per ogni linguaggio riconoscibile da un NFA esiste un corrispondente (più complicato) DFA.

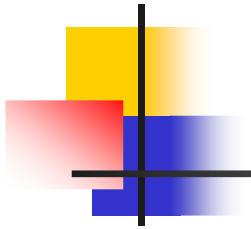


Classificazione
Struttura grammatica
Linguaggi regolari (tipo 3)

Esiste una corrispondenza biunivoca fra linguaggi di tipo 3 (regolari) ed espressioni regolari.

Dato un alfabeto, un'**espressione regolare** su di esso può essere:

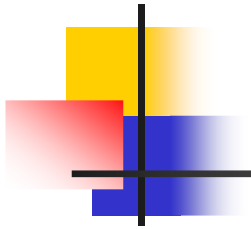
- ε (la stringa vuota)
- $\forall$ simbolo terminale a
- se R è una espressione regolare R* è una espressione regolare
- se R è una espressione regolare R^+ è una espressione regolare. L'operatore + indica una o più ripetizioni
- se R e S sono espressioni regolari, $R \cup S$ (oppure R|S) è una espressione regolare; l'operatore | indica l'alternanza
- se R e S sono espressioni regolari, $R \cdot S$ (oppure RS) è una espressione regolare che descrive la concatenazione di R seguito da S
- [a-z] è una abbreviazione per a|b|c|..|z



Classificazione
Struttura grammatica
Linguaggi regolari (tipo 3)

Le espressioni regolari (ER, o regex) sono molto utilizzate in parecchi linguaggi di programmazione ed anche in comandi di sistema (ad esempio il grep di Unix). Una definizione pratica di ER è "un pattern per descrivere testo", solitamente da ricercare. **Esempi**:

- *a* ricerca la prima occorrenza di "a" entro una data stringa, (solitamente la scansione va da sinistra a destra)
- *.* rappresenta un singolo carattere
- *c[ao]sa* trova le parole "casa" e "cosa", *[^acx]* trova tutto ciò che non è a,c,x
- quantificatori ?,*,+ : *colou?r* trova sia color che colour (? indica 0 o 1 ripetizione dell'ER che lo precede), , *go*gle* trova ggle, gogle, google, gooogle ecc. (* indica 0, 1 o più ripetizioni dell'ER che lo precede), , *go+gle* trova gogle, google, gooogle, ecc. ma non ggle (+ indica 1 o più ripetizioni dell'ER che lo precede). I quantificatori sono *greedy*, ossia tentano di intercettare la massima quantità che loro compete.
- { e } sono usate per indicare quantità specifiche, ad esempio *[1-9][0-9]{3}* trova un numero da 1000 a 9999, *[1-9][0-9]{2,4}* fra 100 e 99999



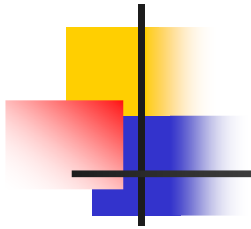
Classificazione
Struttura grammatica
Linguaggi regolari (tipo 3)

- ^ rappresenta l'inizio della riga (*^s* trova le s all'inizio, *^.s* trova la s in seconda posizione), $ rappresenta la fina della riga (*et.$* trova et seguito da un qualsiasi carattere alla fine della riga)
- \ (backslash) seguito da un carattere rappresenta il carattere stesso, ad esempio et\.$ richiede che la stringa trovata sia esattamente "et." alla fine della riga, mentre ..\.\. rappresenta due caratteri qualsiasi seguiti da due punti, .\$\.$ indica la sequenza di un qualsiasi carattere seguito dal dollaro, dal punto, il tutto in conclusione di una riga, \^.\.\$ indica la stringa con ^, un carattere, il punto, ed il dollaro
- \n rappresenta un line feed, \r un ritorno carrello (di fatto, \n rappresenta un ritorno a capo in Unix, mentre in windows si usa \r\n), \t rappresenta una tabulazione; esistono altri simboli per i caratteri non stampabili
- A seconda del linguaggio in cui sono utilizzate, le ER possono essere differenti; quelle con sintassi più estesa sono usate in Perl, Python, .NET

Riferimenti:

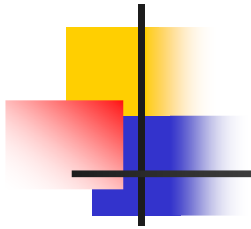
http://www.regular-expressions.info/

http://en.wikipedia.org/wiki/Regular_expression



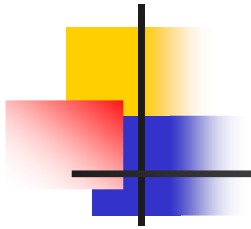
Classificazione
Struttura grammatica
Linguaggi non contestuali (tipo 2)

- Le grammatiche di tipo 2 sono le più importanti nel campo della computer science in quanto quasi tutti i linguaggi di programmazione si basano su grammatiche di questo tipo; sono infatti meno restrittive del tipo 3, ma ancora abbastanza restrittive da assicurare l'efficienza dei relativi traduttori (non troppo complessi)
- Sono dette anche **grammatiche non contestuali** (context-free) perché la sostituzione operata dalle produzioni può essere effettuata sempre, a prescindere dai simboli precedenti o successivi a quelli che costituiscono la produzione stessa (ovvero a prescindere dal contesto)
- una regola context-free del linguaggio naturale potrebbe essere *"this"→"the"* , effettuabile sempre, mentre la regola "the"→"a" è contestuale perché "a" richiede che il simbolo successivo sia una consonante



Classificazione
Struttura grammatica
Linguaggi non contestuali (tipo 2)

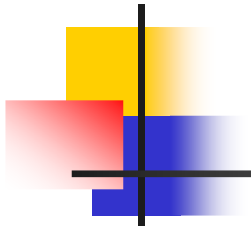
- Il riconoscitore di un linguaggio di tipo 2 è un **automa a pila** (push-down automaton, o PDA), definito come l'insieme di due nastri di lunghezza infinita, uno che rappresenta i simboli di ingresso, ed il secondo usato come stack che impila i simboli in ingresso fin quando riconosce nella sequenza impilata una produzione, al che svuota la pila e prosegue con i successivi simboli di ingresso
- Il PDA, a differenza del FSA, possiede una sorta di "memoria", rappresentata dai simboli già presenti nella pila, che permette il corretto riconoscimento delle produzioni



Classificazione
Struttura grammatica
Linguaggi non contestuali (tipo 2)

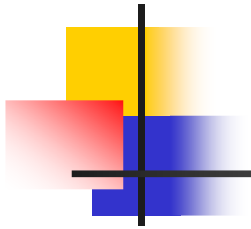
- I linguaggi non contestuali possono essere descritti utilizzando una delle forme normali
- Le **forme normali** sono metodi di descrizione di un linguaggio che permettono di dimostrare il possesso di determinate proprietà; non hanno lo scopo di incrementare la leggibilità delle produzioni
- Le forme normali possono essere usate per tutti i linguaggi, ma sono particolarmente adottate per quelli di tipo 2, che descrivono la maggior parte dei linguaggi di programmazione
- Le forme normali più note sono:
  - **Chomsky normal form** (CNF)
  - **Backus-Naur normal form** (BNF), la più usata, anche nella forma estesa (EBNF), ad esempio un identificatore in Pascal è definito come

    <id> ::= <letter>|<id><letter>|<id><digit>



Classificazione
Struttura grammatica
Linguaggi contestuali (tipo 1)

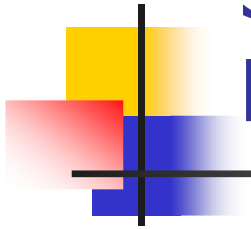
- i linguaggi di tipo 1 sono dipendenti dal contesto;
- il riconoscitore è la **macchina di Turing** con nastro finito, denominata automa linearmente limitato (linear-bounded automaton, o LBA);
- Un LBA è definito come una sestupla:
  - un alfabeto di simboli in ingresso (letti dalla macchina)
  - un alfabeto di simboli in uscita (scritti dalla macchina), non necessariamente coincidente con quello di ingresso
  - un nastro, diviso in celle
  - una testina di lettura/scrittura, che può spostarsi avanti o indietro di una cella sul nastro e che può leggere il simbolo sulla cella posta sotto di essa e modificarlo o cancellarlo



Classificazione
Struttura grammatica
Linguaggi contestuali (tipo 1)

- un insieme finito di stati, inclusi uno di inizio ed uno di fine
- un insieme di regole chiamate *programma*. Ogni regola è del tipo (stato1, read-char, write-char, direzione, stato2), ed indica, dato lo stato corrente (stato1) quale carattere in lettura è ammesso (read-char) e quale carattere scriverà in conseguenza (write-char), muovendo poi il nastro nella direzione indicata (direzione) e portandosi nello stato2.

- la macchina di Turing è particolarmente importante perché è considerata un implementatore universale di algoritmi

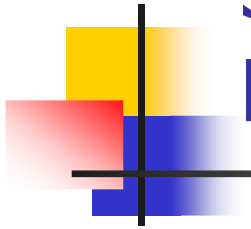


Classificazione
Struttura grammatica
Note storiche – Alan Turing

Alan Turing è il genio matematico che riuscì a decifrare il codice Enigma nel 1942, utilizzato dai tedeschi durante la seconda guerra mondiale, permettendo la vittoria degli alleati anglo-americani.

Nato nel 1912, nel 1933 entrò nell'Università di Cambridge per studiare matematica; in quegli anni la disciplina era alle prese con importanti questioni, relative alla consistenza ed alla completezza della matematica, ed al quesito se vi fosse un metodo "meccanico" (cioè dettato da regole precise) per determinare quali affermazioni siano decidibili, ovvero quali problemi siano risolvibili. Turing affrontò in particolare questa problematica, pervenendo alla definizione di una macchina astratta in grado di risolvere qualsiasi problema seguendo opportune regole.

La promettente carriera di Turing venne stroncata da un'accusa di presunta omosessualità nel 1952, che lo indusse al suicidio nel 1954.



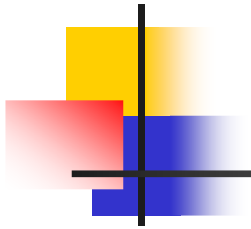
Classificazione
Struttura grammatica
Note storiche – Alan Turing

Nella sua breve vita, Turing si occupò delle questioni più profonde poste dal mondo dell'informatica:

può una macchina essere intelligente quanto e più di un umano? L'emozione e la ragione sono in realtà poi così diverse?

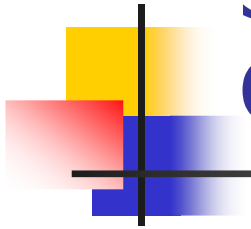
Può una macchina capire le esperienze umane? Amore, frustrazione, morte?

Turing tentò di combinare matematica, filosofia ed ingegneria, una strada considerata solo parecchi anni dopo la sua morte.



Classificazione
Struttura grammatica
Linguaggi liberi (tipo 0)

- i linguaggi di tipo 0 sono i più generali, senza restrizioni sulle produzioni
- i riconoscitori sono ancora macchine di Turing, ma con lunghezza del nastro infinita, così come infinito diventa il numero degli stati
- i linguaggi di tipo 0, detti anche ricorsivamente enumerabili, sono creazioni di forte interesse teorico ma senza significative ripercussioni nel campo dei linguaggi di programmazione



Classificazione
Struttura grammatica
Grammatiche per linguaggi naturali

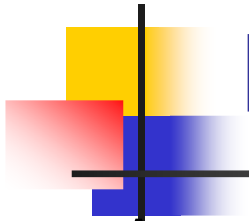
- Il linguaggio naturale (umano) è chiaramente molto complesso, spesso è difficile definire una grammatica (un insieme di regole in grado di generare tutte le possibili frasi)
- Gli studi sulle grammatiche contestuali e non hanno dimostrato la loro utilità nell'analisi dei linguaggi naturali, anche se l'ambiguità e la mancanza di restrizioni rendono difficoltoso lo sviluppo di sistemi per la compresione del linguaggio naturale
- Uno dei metodi più efficaci è l'utilizzo delle **augmented transition network (ATN)**, sviluppate a partire dagli automi ed arricchite con diversi test per determinare l'accordo fra nomi, aggettivi, avverbi, verbi ecc. Le ATN riescono a generare e riconoscere le frasi sintatticamente corrette del linguaggio naturale; resta ancora complessa la comprensione della semantica

Classificazione
Struttura grammatica
Note storiche – Noam Chomsky

Noam Chomsky iniziò lo studio della linguistica a 10 anni, affascinato dagli studi del padre sulle grammatiche del XIII secolo.

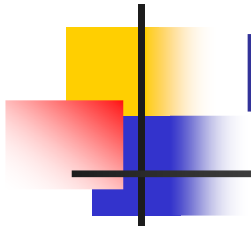
Laureatosi nel 1951 in Pennsylvania, e conseguito il dottorato nel 1955, i suoi studi erano considerati eccessivamente rivoluzionari dalle principali scuole di linguistica, finchè supportò le proprie teorie con solide basi matematiche, ottenendo la prima pubblicazione nel 1957.

Gli interessi di Noam Chomsky non si limitano alla sola linguistica e all'informatica. Intellettuale meticoloso e militante, si avvicina anche al mondo della politica nel 1965, partecipando a movimenti pacifisti in opposizione alla guerra del Vietnam. *The culture of terrorism*, del 1988, critica le posizioni statunitensi nel centro America e in Iran. Seguono altre pubblicazioni razionali ed impietose degli scenari internazionali; ultima in ordine di tempo *Hegemony or survival*, del 2003, raccolta di riflessioni decennali volte a sostenere la tesi del progetto di dominio planetario promosso dagli USA prima e dopo l'11 settembre 2001.



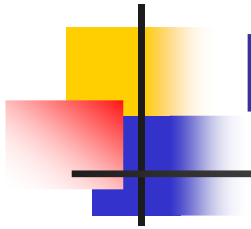
Classificazione
Livello di astrazione

- A causa della struttura interna (basata su transistor utilizzati come switch), i computer possono essere programmati soltanto in **linguaggio macchina (LM)**, sequenza di 0 e 1, sintetizzata talvolta con numeri decimali, ottali o esadecimali, indicata anche come **prima generazione di linguaggi**; nei primi computer scarso era il supporto per la programmazione in linguaggio macchina (pulsanti, schede perforate), che era quindi difficoltosa e soggetta a numerosi errori
- I primi tentativi di facilitare la programmazione portarono negli anni 60 alla nascita dell'**assembly**, **linguaggi della seconda generazione** in cui le istruzioni in LM sono espresse tramite codici alfanumerici mnemonici derivati dalla lingua inglese, introducendo così un primo livello di astrazione, restando ancora tuttavia fortemente dipendente dall'hardware della macchina
- Il passo successivo fu quello di introdurre linguaggi che fossero anche indipendenti dall'hardware, permettendo quindi al programmatore di concentrarsi maggiormente sul problema da risolvere piuttosto che sui dettagli implementativi. Questi linguaggi (C, Pascal, FORTRAN, COBOL, Java ecc.) sono **linguaggi di terza generazione**;



Classificazione
Livello di astrazione

- negli anni 80 sono stati introdotti i **linguaggi di quarta generazione**, che possiedono un ulteriore grado di astrazione, ma che solitamente restringono il campo d'azione rispetto ai linguaggi di terza generazione; un esempio è l'SQL, linguaggio per la gestione di database;
- incrementando il livello di astrazione, con l'obiettivo di utilizzare il linguaggio naturale, si perviene ai **linguaggi di quinta generazione**, attualmente utilizzati nel campo dell'intelligenza artificiale e dei sistemi esperti (che utilizzano motori inferenziali e basi di conoscenza per risolvere problemi)
- i linguaggi 3G, 4G, 5G sono detti anche **linguaggi ad alto livello** perché sono tutti indipendenti dall'hardware della macchina, contrariamente a LM ed assembly, definiti **linguaggi di basso livello**; in generale, un linguaggio è considerato di basso livello se è possibile manipolare la RAM utilizzando istruzioni del linguaggio

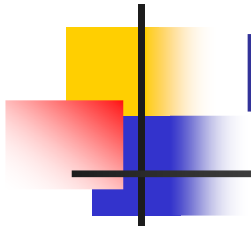


Classificazione
Livello di astrazione

Confronto fra linguaggi ad alto e a basso livello:

	Basso livello	Alto livello
Apprendimento	Difficoltoso	Facile
Uso	Difficoltoso	Facile
Portabilità	Nulla	Buona
Manutenzione programmi	Scarsa	Buona
Efficienza del codice	Alta	Bassa

L'assembly è ancora oggi utilizzato, quando occorre accedere direttamente all'hardware della macchina (sviluppo di driver), quando occorre efficienza, e nei sistemi embedded



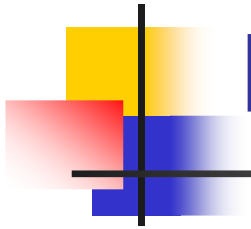
Classificazione
Livello di astrazione

- Qualsiasi sia il linguaggio utilizzato, il computer comprende comunque solo il LM, quindi occorre sempre una traduzione del programma dal linguaggio utilizzato (**codice sorgente**) in LM (**codice oggetto**); solitamente la traduzione avviene con le seguenti fasi:
  - **Analisi lessicale** (scanning), che verifica che i simboli utilizzati nella stesura del sorgente siano ammessi, e li organizza in elementi lessicali composti (token), quali parole chiave ("if", "while"), identificatori (nomi di variabili), costanti numeriche, eliminando anche spazi bianchi e commenti; i token sono definiti solitamente tramite ER
  - **Analisi sintattica** (parsing), in cui si esaminano le sequenze di token per riconoscere il pattern di una regola del linguaggio (costrutto), ad esempio *if <expr> then <cond1> else <cond2>*; le regole sono spesso descritte tramite BNF
  - **Analisi semantica**, in cui si analizza il significato del codice per scoprire eventuali incoerenze (type checking)

Classificazione
Livello di astrazione

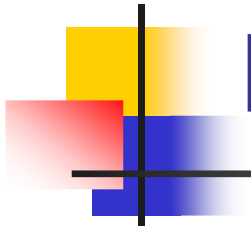
- I traduttori sono di due tipologie: **compilatori** ed **interpreti**
- Il **compilatore** prende in ingresso il codice sorgente e lo traduce completamente in codice oggetto; l'esecuzione del file oggetto è possibile solo a traduzione ultimata
- L'**interprete** traduce ed esegue immediatamente ogni singola istruzione del codice sorgente in codice oggetto, senza produrre un file oggetto distinto
- Confronto compilatore – interprete:

	Compilatore	**Interprete**
Tempo di esecuzione	Basso (viene eseguito codice compilato)	Alto
Correzione errori	Lenta (occorre ricompilare)	Veloce (debug interattivo)
Ottimizzazione del codice	Si	No
Visibilità codice sorgente	No	Si
Necessità del traduttore (occupazione di memoria)	No	Si



Classificazione
Livello di astrazione

- L'utilizzo di linguaggi ad alto livello ha imposto la creazione dei traduttori, però l'astrazione introdotta permette di concentrarsi sul problema da risolvere, demandando ai traduttori il compito di colmare il divario con la macchina sottostante
- I moderni sistemi di elaborazione spesso non presentano solo due livelli (il LM ed il linguaggio ad alto livello), ma introducono livelli intermedi, ognuno con scopi precisi e con l'intento di offrire servizi specifici al livello superiore, per garantire una maggiore astrazione e separazione dei compiti (accesso alle risorse di sistema e/o alle periferiche); ogni livello costituisce una **macchina virtuale**



Classificazione
Paradigma di programmazione

"Universally recognized scientific achievements that for a time provide model problems and solutions to a community of practicnionery"

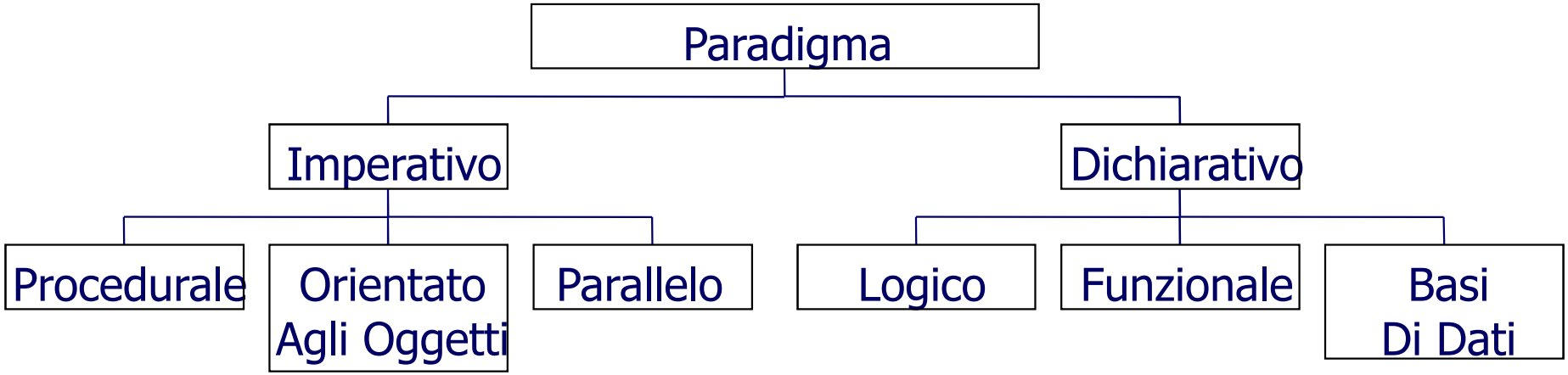
Un paradigma è:

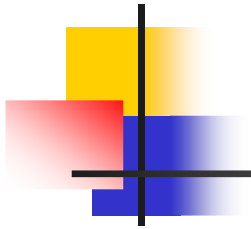
- Un modello di riferimento
- Un insieme di principi di programmazione
- Una raccolta di caratteristiche astratte

La scelta del paradigma che meglio risolve un dato problema è di fondamentale importanza per un programmatore;

alcuni problemi possono essere risolti utilizzando più paradigmi

Classificazione
Paradigma di programmazione

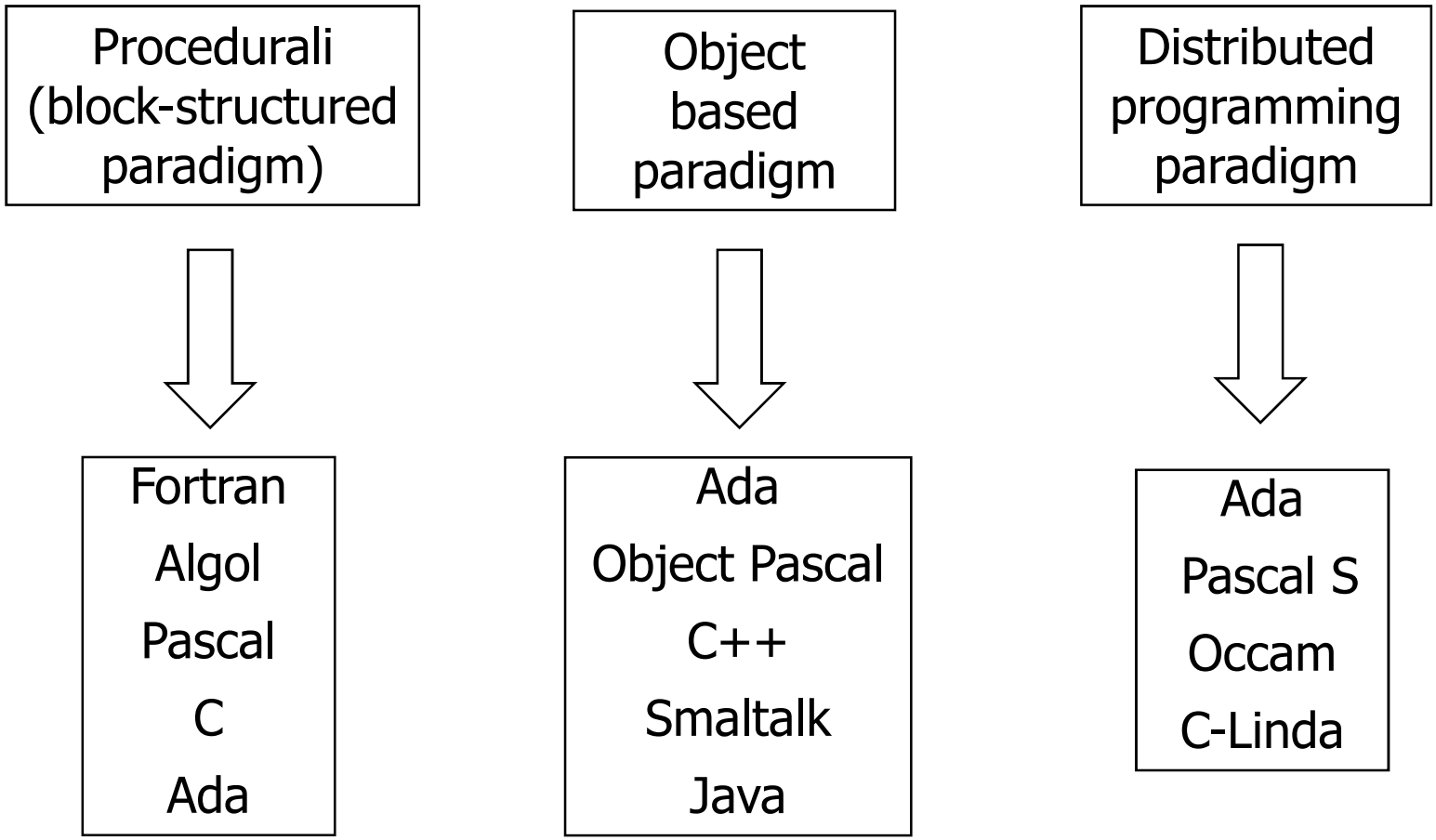


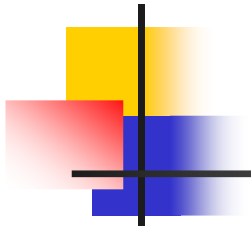


Classificazione
Paradigma di programmazione

- I linguaggi **imperativi** sono quelli in cui il modello di computazione è il cambiamento di stato della memoria
- Sono basati sul concetto di assegnazione di un valore ad una locazione di memoria
- Il compito del programmatore è definire una sequenza di cambiamenti di stato della memoria che produrranno lo stato finale desiderato
- In pratica, nei linguaggi imperativi deve essere il programmatore a specificare "come" va risolto il problema
- I linguaggi imperativi sono divisi nelle tre categorie:
  - **Procedurali**, secondo cui il programma viene organizzato in procedure (funzioni)
  - **Orientati agli oggetti** (Object-oriented), in cui l'elemento di base è l'oggetto, metafora degli oggetti del problema reale
  - **Paralleli**, in cui processi e relativa sincronizzazione sono dominanti

Classificazione
Paradigma di programmazione





Classificazione
Paradigma di programmazione

- I linguaggi dichiarativi sono basati sulla dichiarazione di affermazioni che descrivono una data realtà ed il problema che in tale contesto si desidera risolvere
- Compito del programmatore è utilizzare un linguaggio di specifica per le affermazioni, e fornire le affermazioni opportune per consentire alla macchina di risolvere il problema
- In pratica, nei linguaggi dichiarativi il programmatore si occupa di definire "cosa" vuole, non il come, compito che viene demandato alla macchina (in realtà, è demandato alla capacità inferenziale del traduttore che altri umani hanno scritto per quella macchina)
- L'utilizzo di questi linguaggi è ancora piuttosto ristretto (intelligenza artificiale), fondamentalmente perché i motori inferenziali non sono ancora molto potenti né efficienti

Classificazione Paradigma di programmazione

- I linguaggi dichiarativi sono divisi in tre categorie:
  - Linguaggi logici, in cui si utilizzano assiomi, combinati con le regole della logica, per arrivare al risultato (verità o falsità di un affermazione data come problema)
  - Linguaggi funzionali, in cui sono le funzioni gli elementi di base, ed il problema viene risolto valutandole (fornendo gli argomenti necessari)
  - Database, in cui si utilizzano appositi costrutti (select, insert) per gestire i dati