Introduzione all'Informatica

Corso di Informatica di Base per l'integrazione dell'ultimo anno di Scuola Superiore e per il primo anno di Universitá

Edizioni CEDAM - Padova

Prefazione di ...

Bozze

Matteo Cristani
Dipartimento di Informatica
Università di Verona,
Cà Vignal 2, strada Le Grazie
I-37134 Verona, cristani@sci.univr.it

Prefazione dell'autore

Questa sezione mi occorre per tre motivi. Il primo é che voglio avere l'occasione di ringraziare Marco Rigodanzo e Nicoletta Gabrielli, studenti del corso di Laurea in Informatica dell'Universitá di Verona per avere scritto parte delle definizioni di comando per Linux .

In secondo luogo desidero dedicare questo testo a molte persone: mia moglie che mi ha molto incoraggiato durante la sua stesura; i miei figli che ancora non possono comprendere ció che ho scritto qui e quindi nemmeno correggermi o appuntarmi errori ed espressioni infelici; gli studenti del corso di laurea di Informatica dell'Universitá di Verona che hanno frequentato il corso di Informatica di Base da me tenuto usando le bozze di questo libro e che mi sono serviti da cavie; gli studenti del corso di laurea in Biotecnologie Agro-Industriali dell'Universitá di Verona che hanno seguito il corso di Informatica con lo stesso destino dei loro colleghi informatici.

Colgo anche l'occasione per ringraziare i colleghi che hanno letto ed utilmente commentato le bozze. Voglio anche ringraziare la Casa Editrice CEDAM ed in specie la Dott.ssa Giorato che ha creduto fin dall'inizio in questo ambizioso progetto.

L'autore

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

1.1 Che cos'è l'informatica?

L'affermazione comunemente piú accettata a proposito dell'informatica é che l'informatica é la disciplina che si occupa dei calcolatori. Questo é evidentemente vero, ma tale concezione é comunque fuorviante. L'informatica é, per essere piú precisi, non solo la scienza del calcolatore, ma anche e soprattutto, la *scienza del calcolo*. Ragione di esistenza dell'informatica é:

> la determinazione con metodo sia ipotetico-deduttivo che sperimentale, nonché l'analisi e la definizione di protocolli e metodologie di verifica valide per la costruzione di prototipi e applicazioni ingegnerizzate riguardo a tutto ció in cui consiste la definizione di dati *di ingresso* e la costruzione di soluzioni che comportino la definizione di dati *di uscita*.
>
> *ACM (Association for Computing Machinery)*
> *Syllabus dei corsi universitari di Informatica.*

In questo senso l'informatica é una disciplina non solo indipendente, ma addirittura antecedente rispetto alla costruzione di calcolatori elettronici. Rimane il fatto che, una volta eseguita la costruzione di sistemi adatti per l'esecuzione di calcoli, l'informatica come disciplina autonoma ricevette una fenomenale spinta in avanti, non solo *guidata* dall'elettronica, ma in certe fasi, perfino *come guida* per l'elettronica.

In questo primo capitolo faremo una rapida e pragmatica carrellata storica relativa all'evoluzione della tecnologia informatica e della relativa disciplina teorica, l'informatica appunto.

1.2 Elaborazione meccanica dell'informazione: cenni storici

I tre protoantecedenti dei moderni sistemi di elaborazione dell'informazione vennero concepiti nella prima metá del XIX secolo e vennero perfezionati nella seconda metá.

L'idea di un macchiario in grado di svolgere complessi compiti non é certamente risultato della produzione intellettuale del solo XVIII secolo, essendo in effetti frutto di secoli di sviluppo tecnologico. Va peró riconosciuto che, senza alcuna ombra di dubbio, gli arabi dominatori della scena culturale europea e magrebina dei secoli dal IX al XIII ebbero un ruolo eminente nello sviluppo di quelle concezioni, che come vedremo, sarebbero state la base della costruzione di un modello applicabile di calcolo automatico.

Le prime macchine con caratteristiche del tipo di cui sopra erano sistemi di automazione industriale, in particolare per l'industria tessile. Il piú famoso di tali modelli é noto con il nome di telaio Jacquard, che era controllato da schede perforate di cartone. Con queste schede si rendeva automatica la lavorazione della stoffa e i disegni realizzati nello stabilimento di tessitura del padre. Le schede furono lo spunto per la programmazione della calcolatrice meccanica ideata da Charles Babbage. In Figura 1.1 presentiamo un disegno di uno dei primi telai meccanici dello scienziato francese.

Due macchinari per il calcolo vennero invece costruiti con scopi accademici. Per maggior precisione occorre dire che in un caso, per il filosofo Pascal, la macchina per fare calcoli che egli progettó, venne effettivamente costruita, mentre l'altro dispositivo, noto con il nome di macchina analitica, concepito dell'eclettico ingegnere e inventore inglese Charles Babbage non ebbe mai luce fino a poco tempo fa, quando venne costruita a partire dai disegni di Babbage al museo della scienza e della tecnica di Milano. Alcuni infruttuosi tentativi erano stati effettuati in precedenza. In Figura 1.2 compare una fotografia di una moderna realizzazione della macchina analitica.

Nella Figura 1.3 compare una fotografia della macchina calcolatrice di Pascal.

1.3 Il calcolatore elettronico ed il suo sviluppo

La genesi storica della moderna informatica é, come abbiamo accennato nella sezione precedente, un lungo processo che affonda le proprie radici sia nella matematica e nella logica che nell'Ingegneria Elettronica.

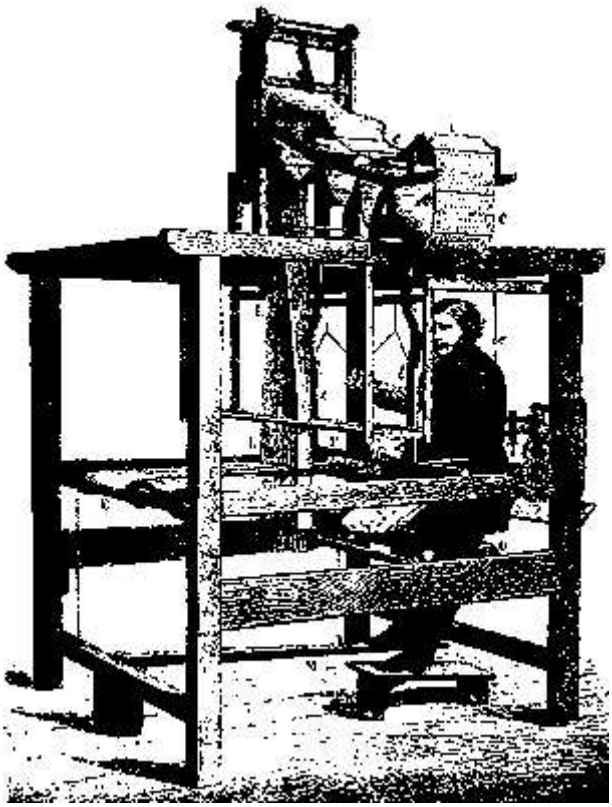

Figura 1.1: Un disegno di uno dei primi telai Jacquard a schede perforate

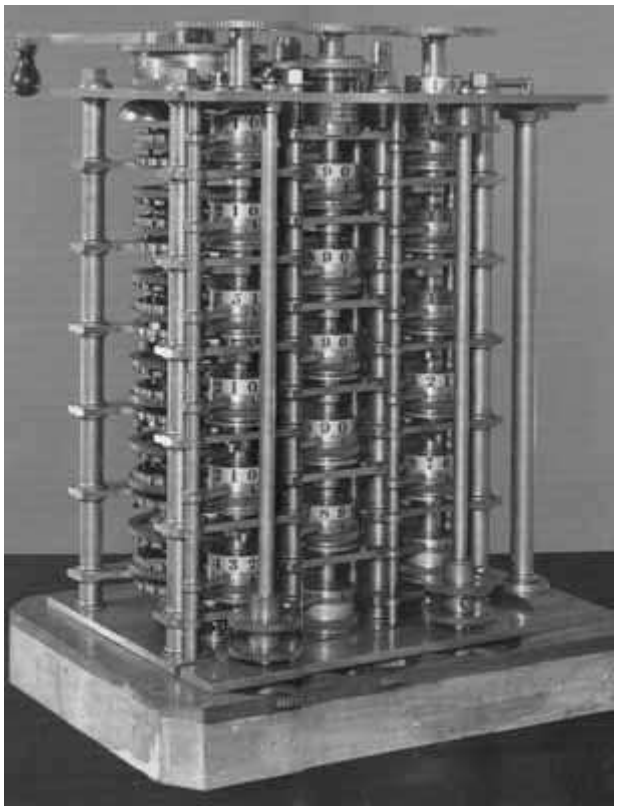

Figura 1.2: Una fotografia di uno dei progetti di ricostruzione della macchina analitica di Babbage

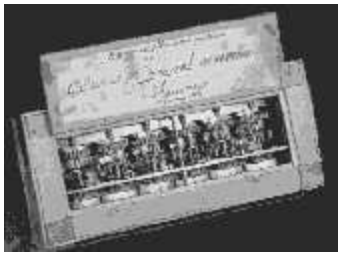

Figura 1.3: Una fotografia della macchina calcolatrice di Pascal

Come spesso avviene nel corso dello sviluppo storico, la nascita di una nuova tecnologia si fonda sulla determinazione di una motivazione pratica che ne sia motore. Nel caso dell'Informatica a far da motore furono, in origine, motivazioni di carattere bellico. Durante la Seconda Guerra Mondiale, un grande sforzo venne condotto, da parte del governo statunitense, per promuovere la costruzione di un ordigno atomico, che allora si pensava competere con un ipotetico ordigno atomico tedesco, che era in effetti rimasto sulla carta. Lo sviluppo di tale progetto, il Progetto Manhattan, si compícon l'aiuto di svariate competenza, tra cui emergeva, evidentemente, quella dei fisici. Molti dei calcoli necessari per la determinazione dei parametri critici in tale progetto venivano calcolati, come usava al tempo, mediante l'aiuto di un cospicuo numero di *ingegneri calcolatori*, cioé esperti in esecuzione di calcoli complessi. Il numero di persone impiegate in ogni singolo calcolo dipendeva, evidentemente, dalla complessitá del calcolo stesso. Gli esiti, per quanto positivi, erano sconfortanti in termini di tempo impiegato e di risorse umane necessarie. Le modalitá operative erano basate, principalmente, sulla disposizione in lunghe file degli ingegneri calcolatori, che venivano assoggettati al controllo di alcuni *coordinatori del calcolo* che facevano passare le informazioni necessarie per posta pneumatica.

Nel frattempo la letteratura ingegneristica si popolava di progetti di calcolatrici elletromeccaniche. Tre ricercatori tentarono la costruzione di vere e proprie macchine per il calcolo, ispirate al modello astratto noto con il nome di Macchina di Turing, progettata dal matematico Inglese Alan Turing nel 1936.

Nel 1936 Konrad Zuse costru nel salotto di casa lo Z1 , una macchina da calcolo che sfruttava la tecnologia elettromeccanica del rel[1]. Nel 1941, presso i laboratori del politecnico di Berlino costruisce lo stesso Zuse realizzó lo Z3.

Howard Aiken nel 1944 alluniversitá di Harvard, terminó la realizzazione di MARK 1, una macchina elettromeccanica relativamente veloce, realizzata in soli 5 anni per gli stessi motivi bellici che ispirarono i progetti di computer legati al progetto Manhattan. É ancora elettromeccanica , lunghezza 15 m e altezza 2,5 metri. Una fotografia di Mark1 appare in Figura 1.4.

Stilbitz dei Laboratori di ricerca della Bell Telephone usó relé per ottenere un computer "miniaturizzato", cioé che stesse in una stanza mediopiccola.

Il modello di esecuzione dei calcoli degli ingegneri calcolatori, insieme ad altre proposte della corrente letteratura di ingegneria, in particolare la

[1]Circuiti elettrici con interruttori elettromeccanici

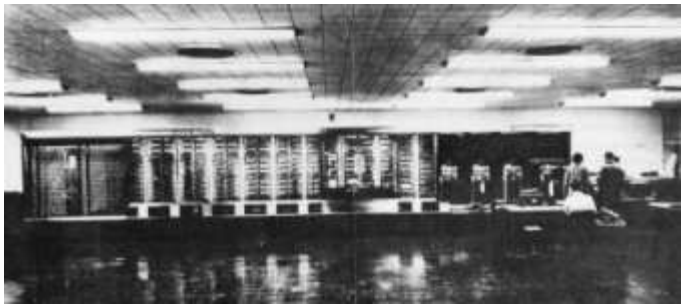

Figura 1.4: Mark1: uno dei primi calcolatori elettromeccanici a relé.

letteratura sui servomeccanismi, noncé le macchine elettromeccaniche per il calcolo, e lo sviluppo di alcune moderne tecnologie elettroniche, come le valvole termoioniche, ispiró una proposta di modello di un ipotetico calcolatore elettronico, il cui primo prototipo venne effettivamente costruito nel 1942, ad opera dell'ingegnere di origine rumena e russa, la "macchina per il computo elettronico" ABC (Satanasso-Berry-Computer). Il calcolatore di Satanasso fu costruito insieme al suo allievo Berry ;la memoria del calcolatore era costituita da condensatori fissati ad un grande tamburo cilindrico; ogni tamburo conteneva 1500 bit.

Il progetto legato al progetto Manhattan, e guidato dal fisico ungherese Von Neumann, non venne mai completamente realizzato. Quello che si considera il primo vero calcolatore elettronico nacque in ambiente accademico. . Nel 1946 Eckert e Mauckly nelluniversitá della Pennsylvania costruiscono lENIAC (Electronic Numerical Integrator and Computer) che fu ritenuto fino al 1973 il primo calcolatore elettronico programmabile. Era una colossale macchina costituita da 30 armadi alti 3 metri ciascuno: occupava unintera stanza, infatti il suo peso era di 30 tonnellate ed occupava una superficie di 180 mq ed era in grado di eseguire 300 moltiplicazioni al secondo. Era elettronico con un sistema a valvole e la programmazione avveniva azionando manualmente degli interruttori. In Figura 1.5 compare una fotografia di ENIAC.

Intanto nell'Inghilterra di W. Churchill si portava a termine nel 1943 l'elaboratore dedicato Colossus, rimasto segreto sino al 1970. In Figura 1.6 compare un'immagine di tale calcolatore.

La tecnologia elettronica alla base di tale sistema erano le cosiddette valvole termoioniche, cioé sistemi di memorizzazione di dati in aritmetica binaria basati sulla ionizzazione termica di un gas. Una valvola termoionica é

Figura 1.5: Una fotografia del calcolatore elettronico ENIAC.

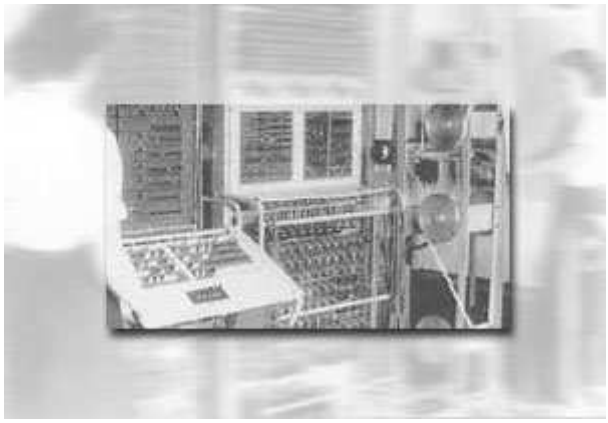

Figura 1.6: Una fotografia di Colossus, calcolatore Inglese realizzato segretamente nel 1943.

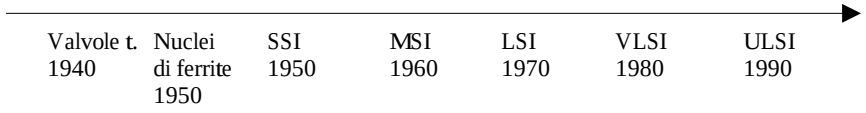

Figura 1.7: Evoluzione delle tecnologie per il calcolo.

un oggetto ingombrante, che nella migliore delle ipotesi di miniaturizzazione raggiunge le dimensioni di alcuni centimetri.

Per scendere sotto il centimetro occorse passare ad una tecnologia di memorizzazione basata sul magnetismo, i cosiddetti *nuclei di ferrite*. Anche questa tecnologia difettava, in specie, per la irriducibilitá delle dimensioni.

Il passaggio fondamentale, la *rivoluzione elettronica* che consentídi ottenere calcolatori le cui dimensioni potevano essere gestite senza particolari accorgimenti di impianto[2] avvenne grazie ad un innovativo dispositivo elettronico: il transistor. I transistor sono dispositivi elettronici basati sul comportamento ambivalente dielettrico/conduttore dei materiali noti con il nome di semiconduttori, ed in particolare, il silicio. Perché la tecnologia del silicio ha avuto il poderoso successo che sta sotto gli occhi di tutti? Principalmente perché in misura crescente é stato possibile sfruttare basi di silicio a forma di sottile fetta, note con il nome di *wafer* per *integrare* transistor di dimensione molto piccola. I circuiti cosídefiniti sono noti con il nome di *circuiti integrati*. Le scale di integrazione sono state poi classificate da SSI (Small Scale Integration) fino a ULSI (Ultra Large Scale Integration) per i piú moderni circuiti. Le sigle intermedie MSI, LSI, VLSI (Medium, Large e Very Large Scale Integrations) corrisposero a vari stadi dell'integrazione a partire dall'integrazione di qualche centinaio di circuiti negli SSI, qualche migliaio nei MSI e qualche centinaio di migliaia per i LSI, fino ai milioni per i VLSI. La corrente generazione ULSI prevede che nello stesso wafer si integrino diversi milioni di circuiti.

In Figura 1.3 mostriamo uno schema dell'evoluzione delle tecnologie di base per il calcolo.

[2]Raffreddamento, alimentazione e la stessa disposizione spaziale erano obiettivamente complicate con le tecnologie termoionica e dei nuclei di ferrite.

1.4 L'evoluzione dei sistemi di elaborazione dell'informazione

Storicamente, a parte gli antecedenti di cui abbiamo avuto modo di parlare in Sezione 1.3 e nell'Introduzione, i sistemi di elaborazione dell'informazione hanno subito una enorme propulsione dallo sviluppo di tecnologie hardware. Indubitabilmente, peraltro, lo sviluppo dell'informatica ha seguito un andamento che si é enormemente modificato negli ultimi quindici anni. Il *gap tecnologico* tra hardware e software, quella differenza negativa che fa síche sistemi operativi ed applicazioni siano relativamente pesanti rispetto all'hardware su cui vengono eseguiti, é frutto di una rincorsa tra evoluzioni.

Le tappe fondamentali sono correlate alle tecnologie di interfaccia. A partire da input per regolazione (manopole) ed output su lampade (i led), i sistemi hanno raggiunto una condizione in cui sia l'input che l'output sono multimediali (video touchscreen, mouse, pad, joystick, casse). La dimensione dell'evoluzione é sotto gli occhi di tutti, oltre alla multimedialitá, sulla Rete Mondiale di Internet.

1.5 Evoluzione storica dei sistemi operativi

L'evoluzione dei sistemi operativi, in particolare, ha seguito l'andamento dello sviluppo dell'hardware, sia per quel che riguarda la struttura dell'interfaccia con l'utente, sia per quel che riguarda la tipizzazione delle periferiche. In specie il cambiamento piú significativo degli ultimi anni sta proprio nell'introduzione della nozione di *plug and play*, un meccanismo che consente di aggiungere una periferica, al limite senza che il tipo della stessa venfa specificato dall'utente, mentre il sistema operativo ospite é attivo.

Il cambiamento piú eclatante sta peró nell'evoluzionme di sistemi home (i personal computer) a partire dagli anni settanta, fino ai portatili attuali le cui dimensioni rappresentano un traguardo straordinario sul piano della mobilitá informatica. Analogamente l'evoluzione di dispositivi innovativi, come il palmare, ha indotto lo sviluppo di sistemi operativi differenziati ed aperti. In questo senso é interessante verificare la compatibiliá di sistemi su palmare come Pocket PC e Palm OS con le tecnologie di sistema operativo complete come Windows 2000, o Windows ME.

Un passaggio cruciale che distingue nettamente generazioni di calcolatori é definito dalla dimensione delle *parole macchina* che il sistema gestisce. I primi sistemi operativi lavoravano a 4 bit, quelli attuali lavorano a 32 o 64. Per lungo tempo, parlando, ad esempio, del mercato dei Personal Computer,

hanno convissuto hardware a 32 bit (dal processore Intel 80386 in avanti) con sistemi operativi che gestivano solo i 16 bit (fino a Windows 95). La differenza tra sistemi operativi con parole a 16 bit e quelli con parole a 32 consiste nella possibilitá di usare nomi "lunghi" ovvero nomi che possono essere fatti di un numero massimo di 128 inclusa l'estensione (contro gli 8 + 3 per l'estensione dei 16 bit) e possono includere quasi tutti i caratteri del codice ASCII, in particolare incluso lo spazio.

1.6 Che cos'è l'informatica teorica

L'informatica teorica ha ricevuto la seguente definizione:

> L'informatica teorica é la disciplina che studia il calcolare come oggetto matematico.

In linea di principio, quindi, l'informatica teorica é disgiunta dalla disciplina informatica in senso stretto che si occupa invece della produzione di programmi, ovvero del calcolo come tecnica. Viceversa, molte delle problematiche astratte studiate in via teorica possono condurre allo sviluppo di utili tecniche di risoluzione di problemi per mezzo di programmi . La qualitá del software dipende in larga misura dalla conoscenza del problema posseduta da coloro i quali sviluppano il sotware stesso.

1.7 Piano dell'opera

Questo libro é organizzato in cinque parti: Teoria, Laboratorio, Esercizi teorici, Esercizi applicati, Autovalutazione. Nella prima parte i temi introdotti sono le Architetture dei Sistemi, la Teoria degli automi, i Sistemi Operativi, la Teoria degli algoritmi. Nella parte di Laboratorio, viceversa, vengono introdotti i fondamenti pratici di uso del sistema operativo Linux e del sistema Windows, nonchè le basi dell'uso di strumenti di produttivitá individuale.

Dopo il presente capitolo di introduzione e prima delle cinque parti su citate in un capitolo sintetico illustreremo in modo completo la bibliografia di riferimento, in modo da evitare che i riferimenti bibliografici appesantiscano la lettura deli capitoli di teoria e di laboratorio.

Capitolo 2

Bibliografia ragionata

Premessa

Questa bibliografia ragionata descrive alcuni testi ma non tutti quelli, inclusi gli articoli scientifici, che appaiono nelle referenze bibliografiche. L'obiettivo é quello di fornire un riferimento ampio e generale, ma non esaustivo.

Questo libro di testo é stato costruito utilizzando il linguaggio TEX, il cui testo di riferimento é [20].

2.1 Bibliografia riferita all'Introduzione

Per quel che riguarda la terminologia generale riferita all'informatica di cui si fa cenno nell'Introduzione, un riferimento di ottima qualitá é il Dizionario Informatico[22], pubblicazione elettronica curata da Francesco Longo dell'Universitá di Venezia e disponibile on–line su

http://www.dsi.unive.it/ flongo/diz.

Altri riferimenti on–line sono:

- Il dizionario elettronico dell'universitá di Pisa [1];
- L'enciclopedia informatica on–line **PC webopaedia** [30];
- I dizionari ricercabili (in inglese) FOLDOC [2], OneLook [38], WhatIs [39] e AcronymFinder [35];
- Le risorse documentali sull'Informatica della Commissione Europea Open Information Interchange [12];

- I glossari dei siti www.incoming.com [37] e www.geek.com [36];

Per un riferimento generale all'informatica alcuni testi di generale consultazione sono [24, 42, 40, 18].

Un secondo aspetto trattato nell'Introduzione é la Storia della disciplina Informatica. Per un riferimento generale si puó consultare il sito

http://www.museoscienza.org/computer/

reperibile in bibliografia [14].

2.2 Bibliografia su architettura dei sistemi di elaborazione dell'informazione

I testi di riferimento generali che indichiamo sono tre:

- Il Patterson[7], un testo orientato alla descrizione delle modalitá con cui effettuare il Progetto Digitale [7];
- Il Tannenbaum[28], piú vicino i metodi della fisica applicata;
- Il recente testo di Fummi et al.[15], molto completo sia dal punto di vista elementare che sul piano delle problematiche piú avanzate.

Sull'architettura delle reti di calcolatori, in particolare, due testi hanno un certo rilievo, per gli scopi di questa bibliografia:

- Uno dei vari testi di Tanenbaum [27];
- Un testo di Damiani, su Internet e le sue tecnologie [13].

Un testo piú tecnico e vicino a specifiche problematiche elettroniche, come la dissipazione di potenza é il libro di Heat ed Halladay [16]. Un testo analogo ma riferito all'architettura delle reti, anche se un po' datato dal punto di vista tecnologico é il Yakubaitis [41].

Infine un aspetto importante é quello delle architetture dei sistemi usati per la gestione di dati a livello professionale (il cosiddetto software gestionale) [11].

2.3 Bibliografia sulla Teoria della Computazione

Un testo di generale riferimento per gli argomenti di Informatica Teorica ed in particolare della teoria dei linguaggi e della computazione é il classico di Aho Sethi e Ullmann [5]. Un altro testo di grande rilievo anche storicamente, é il libro di Aho, Hopcroft e Ullmann [4]. Un testo piú complesso, ma anche molto significativo é il Mandrioli-Ghezzi, seppure un po' datato [23].

In specie riferiamo alcuni lavori che riguardano gli automi a stati finiti e quelli a stati finiti deterministici [9, 10].

2.4 Bibliografia sui Sistemi Operativi

2.5 Bibliografia sulla Programmazione e la Teoria degli Algoritmi

Sue testi molto significativi sono il testo di Hopcroft [17] e quello di Sedgewick [25]. Molti altri testi esistono in letteratura, ma la loro lettura é parecchio ostica e non da consigliarsi a studenti digiuni di altri piú approfonditi argomenti di informatica.

Un buon testo che introduce all'intera famiglia di linguaggi moderni (C, C++, Java, Java2) da un punto di vista molto generale é il libro di Kerninghan e Ritchie [19].

2.6 Bibliografia sul Sistema Operativo Linux

Il piú significativo riferimento íl sito degli utenti di Linux, a cui fanno da ponte un numero molto elevato di riferimenti ulteriori:

http://www.linux.org/

che si puó trovare in letteratura [3].

2.7 Bibliografia sul Sistema Operativo Windows

Analogamente a quanto suggerito nella Sezione 2.6 per Windows un buon riferimento é la rete degli utenti Microsoft

http://www.msn.it/

che si puó trovare in letteratura [29]. Inoltre la guida in linea di Windows é una ragionevole lettura iniziale.

2.8 Codici e standard

Le codifiche utilizzate in questo testo sono:

- Le codifiche standard per gli indirizzi Internet [26];
- Le codifiche standard per i codici internet dei paesi [6];
- Le codifiche per la rappresentazione dei nomi delle lingue [31];
- Le codifiche per la rappresentazione dei nomi dei paesi [34];
- La codifica dei caratteri UCS - ANSI standard [32];
- La codifica UNICODE [33].

Parte I

Informatica di base - teoria

Capitolo 3

Elementi di architettura dei sistemi di elaborazione dell'informazione

3.1 Introduzione

Lo scopo del presente capitolo non é l'introduzione tecnica degli argomenti correlati con la costruzione di sistemi informatici dal punto di vista dell'hardware. Molto poco verrá speso in questo momento per ottenere motivazioni teoriche di inquadramento. In effetti in un corso introduttivo di Informatica l'obiettivo non é di approfondire i temi discussi ma di definirne le caratteristiche generali. Gli obiettivi del presente capitolo sono quindi:

- La definizione delle architetture degli elaboratori elettronici;
- La strutturazione dei metodi di calcolo dell'aritmetica in base 2;
- Lo studio dei circuiti digitali

3.2 Elaborazione dell'informazione e modelli del calcolatore

I modelli di calcolo attualmente ingegnerizzati sul mercato o per i quali esistono prototipi significativi sono fondamentalmente tre

- Le architetture tradizionali;

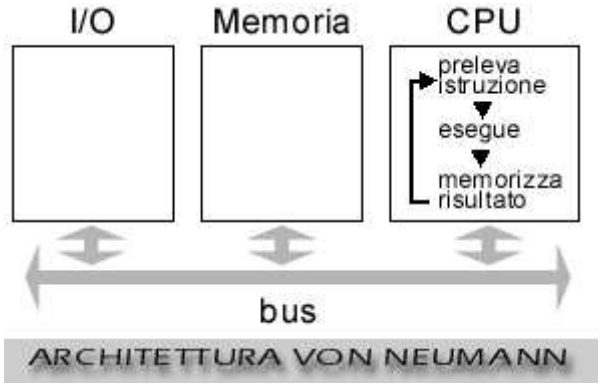

Figura 3.1: Schema generale dell'architettura di Von Neumann.

- Le architetture parallele;
- Le architetture neurali.

Le architetture tradizionali, note con il nome di architetture di Van Neumann, verranno descritte nel paragrafo 3.3. In questo testo le architetture parallele e neurali non saranno discusse.

3.3 L'architettura di Von Neumann

L'architettura di Von Neumann é un modello di struttura funzionale del calcolo. Si tratta percó di uno *schema ingegneristico* che puó essere usato per comprendere il funzionamento di un elaboratore elettronico.

Esso si basa sull'idea che per effettuare elaborazione occorre vedere il calcolatore come un sistema dotato di Input ed Output (I/O), di una componente di elaborazione dell'informazione (CPU - Central Processing Unit), di un sistema di registrazione e conservazione dei dati (Memorie) e di un sistema per lo scambio di informazioni tra le suddette componenti (Bus). Si considerano centrali la CPU, la Memoria ed il Bus. Si considerano periferici i sistemi di I/O.

Lo schema generale dell'architettura di Von Neumann appare in Figura 3.3

Von Neumann, John (Jansci)(1903-1957) nel 1944 divenne consulente nell'Universita' di Pennsyvania del gruppo per lo sviluppo del primo computer elettronico, l'ENIAC. Nel 1945, insieme a John Eckert, John Mauchly, Arthur Burks ed Herman Goldstine, definíl'architettura logica dell'elaboratore. Dimostro' che una macchina con tale architettura (l'attuale computer) poteva eseguire qualunque calcolo risolubile. Da allora l'architettura (o modello) dei computer e' rimasta la stessa; si noti infatti qui in basso la similitudine con i componenti attuali, messi tra parentesi.

La struttura di una macchina binaria costituita da:

- l'unita' aritmetica centrale, un componente che si occupi dei calcoli -operazioni logiche- (ALU);
- l'unita di controllo che gestisce un numero finito di istruzioni in memoria e di dati (DATAPATH);
- la memoria che contenga dati e istruzioni (REGISTRI e RAM)
- la memoria volatile (una delle innovazioni introdotte dal modello di Von Neumann);
- le unitá di ingresso e di uscita (I/O, monitor, tastiera e la memoria di massa, ecc..).

Nel 1946 all'Institute of Advanced Study (Princeton - USA) Von Neumann introdusse il concetto della *elaborazione a programma immagazzinato*, cioé della macchina di uso generale che di volta in volta svolge un lavoro diverso in relazione alla sequenza di istruzioni immagazzinata in memoria. Il primo calcolatore elettronico con queste caratteristiche fu EDVAC.

I moderni dispositivi periferici possono essere classificati come dispositivi di solo Input, di solo Output od entrambi.

Nella Tabella 3.1 compare un elenco delle periferiche piú in uso al momento dell'estensione di questo testo.

Il funzionamento dell'unitá centrale di elaborazione é definito da un ciclo operativo descritto nel riquadro corrispondente. Tale ciclo viene gestito in una sequenza di lettura dell'istruzione, esecuzione e ripetizione. Perché questo meccanismo funzioni occorre operare in modo sincronico, cioé mediante un operatore che induca l'unitá a prelevare periodicamente l'istruzione stessa (il clock), eseguirla in termini operativi mediante una unitá aritmeticologica (la ALU) sotto il controllo della unitá di controllo (la CU) il cui scopo é gestire la lettura e la scrittura di dati e risultati del calcolo nelle memorie di servizio dell'unitá centrale di elaborazione (i registri).

Nome periferica	**Tipo periferica**
Dischi rigidi	I/O
Dischi floppy	I/O
Dischi CD-ROM	Input
Dischi DVD-ROM	Input
Masterizzatori CD-RW	I/O
Masterizzatori DVD-RW	I/O
Stampanti	Output
Scanner	Input
Video	Output
Mouse	Input
Tastiera	Input
Casse audio	Output
Microfono	Input
WebCam	Input
Modem	I/O

Tabella 3.1: Elenco di periferiche in uso.

La struttura dell'unitá centrale di elaborazione é specificata in Figura 3.3.

3.4 Codifica binaria dell'informazione ed aritmetica in base due

§ Conversioni

Operare algebricamente in un sistema diverso dai sistemi in base dieci potrebbe risultare in un primo momento complesso. Ricordiamo qui le regole di conversione.

Regola 1 (Conversione da base 10 a base 2.)

Sia x il numero in base 10 rappresentato di cui vogliamo calcolare la rappresentazione in base 2. Ovviamente la cifra binaria piú significativa di un numero in base 2 é 1.

Sia 2^n la piú grande potenza del 2 minore od uguale a x. Il numero avrá $n+1$ cifre (binarie) nella rappresentazione in base 2. Per computare le cifre

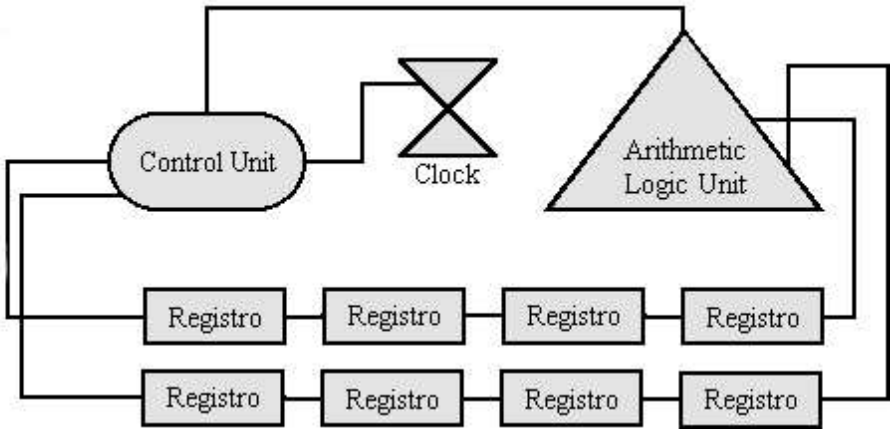

Figura 3.2: Schema dell'unitá centrale di elaborazione.

dalla n-esima alla prima procediamo con il seguente metodo. Calcoliamo x_n come la differenza tra x e 2^n. Calcoliamo poi successivamente x_i come segue

$$x_i = \begin{cases} x_i & se \;\; x_i < 2^i, \\ x_i - 2^i & altrimenti \end{cases}$$

Ora calcoliamo le cifre del numero convertito secondo lo schema per cui ove compare un $x_i = 0$ l'i-esima cifra sará 0, mentre ove compare $x_i \neq 0$ la i-esima cifra sará 1.

■

Ad esempio, volendo convertire il numero 119 decimale in binario dobbiamo procedere come segue:

- La piú grande potenza del due inferiore a 119 é $64 = 2^6$, dunque il risultante numero binario avrá 7 cifre binarie.
- La differenza iniziale $x_6 = 119 - 64 = 55$,
- La sesta cifra binaria si ottiene considerando 2^5 che é minore di 55, perció risultando in una cifra 1; $x_5 = 55 - 32 = 23$;
- La quinta cifra é 1. $x_4 = 23 - 16 = 7$;
- La quarta cifra é 0 percheé $7 < 8$. $x_4 = 7$;

+	0	1
0	0	1
1	1	10

Tabella 3.2: Somma Binaria di due numeri ad una cifra

·	0	1
0	0	0
1	0	1

Tabella 3.3: Prodotto Binario di due numeri ad una cifra

- La terza cifra é 1. $x_3 = 7 - 4 = 3$;
- La seconda cifra é 1. $x_2 = 3 - 2 = 1$;
- La prima cifra é 1.

La codifica risultante é 1110111, che infatti produce $1 * 2^6 + 1 * 2^5 + 1 * 2^4 + 0 * 2^3 + 1 * 2^2 + 1 * 2^1 + 1 * 2^0 = 64 + 32 + 16 + 4 + 2 + 1 = 119$

Il processo di conversione inverso é definito secondo la ovvia interpretazione decimale dell'espressione binaria letta nel modo che abbiamo descritto informalmente per la riprova della risultante corretta codifica binaria.

§ Somma e prodotto di numeri binari

La somma di due numeri binari si ottiene considerando la tabella 3.2.

Viceversa, il prodotto, che é molto piúy semplice da trattare é computato in Tabella 3.3.

Per esempio volendo eseguire la somma di 13 e 21 decimali (pari ovviamente a 34) dobbiamo procedere per prima cosa alla conversione in base 2 dei due numeri. In particolare risulterá 13 = 1101, e 21 = 10101. Per effettuare la somma procediamo dunque come si fa con i numeri decimali incolonnando in corrispondenze posizionali.

	0	1	1	0	1	+
	1	0	1	0	1	=
					0(1)	
				1	0	
			0(1)	1	0	
	0(1)	0	0	1	0	
1	0	0	0	1	0	

Nel caso del prodotto le cose sono molto piú semplici in tabella, ma dato che l'operazione richiede somme successive abbiamo gli stessi problemi della somma.

Per esempio si supponga di voler eseguire il prodotto dei numeri 7 e 6, che in base 2 risultano rispettivamente 111 e 110.

			1	1	1	×
			1	1	0	=
			0	0	0	
		1	1	1	-	
	1	1	1	-	-	
1	0	1	0	1	0	

3.5 Circuiti digitali e porte logiche

Chiamiamo *porta logica* un dispositivo elettronico capace di trasformare la tensione posta ai capi di fili in ingresso nella tensione di fili in uscita dal dispositivo, in modo da eseguire o una conservazione (filo con tensione in ingresso per filo con tensione in uscita e una inversione (filo senza tensione in ingresso per filo con tensione in uscita. Normalmente tali dispositivi vengono realizzati con due tipi di filtro:

- I filtri passa basso: filtri che lasciano passare corrente pari ad una *soglia*, se la tensione all'ingresso é inferiore alla soglia. Questo tipo di filtro richiede chiaramente alimentazione e, in termini elettrotecnici, *dissipa potenza* ovvero produce surriscaldamento;

- I filtri passa alto: filtri che, al contrario dei passa basso, lasciano passare tensione solamente se la tensione d'ingresso supera la soglia. In questo caso non compare alimentazione e conseguentemente nessun effetto di dissipazione e conseguente surriscaldamento.

In Figura 3.5 mostriamo il funzionamento dei filtri bassa basso (a) e passa alto (b).

Le porte logiche possono quindi essere descritte astrattamente in base a come filtrano gli ingressi sulle uscite. Porte con un solo ingresso ne esistono ovviamente solo due, di cui una non ha alcuna utilitá pratica, corrispondendo di fatto all'assenza di filtraggio. Con due ingressi esistono ovviamente le due porte corrispondenti ai connettivi $\wedge$ (porta AND) e $\vee$ (porta OR), nonché le porte corrispondenti alla negazione di AND (porta NAND), alla negazione

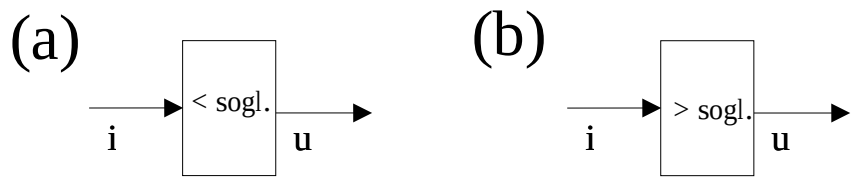

Figura 3.3: Tipi di filtro.

A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

Tabella 3.4: Tabella di veritá del connettivo $\wedge$ (o della porta AND)

di OR (porta NOR) e la porta corrispondente alla disgiunzione esclusiva (porta XOR).

Il comportamento di ciascuna delle su elencate porte é descritto rispettivamente nella Tabella 3.4 per l'AND, nella Tabella 3.5 per l'OR, nella Tabella 3.6 per il NAND, nella Tabella 3.7 per il NOR e nella Tabella 3.8 per lo XOR.

In Figura 3.5 elenchiamo gli schemi grafici corrispondenti alle suddette porte.

A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

Tabella 3.5: Tabella di veritá del connettivo $\vee$ (o della porta OR)

A	B	A NAND B
0	0	1
0	1	1
1	0	1
1	1	0

Tabella 3.6: Tabella di veritá del connettivo $\neg(A \wedge B)$ (o della porta NAND)

A	B	A NOR B
0	0	1
0	1	0
1	0	0
1	1	0

Tabella 3.7: Tabella di veritá del connettivo $\neg(A \vee B)$ (o della porta NOR)

A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Tabella 3.8: Tabella di veritá del connettivo $(A \wedge \neg B) \vee (\neg A \wedge B)$ (o della porta XOR)

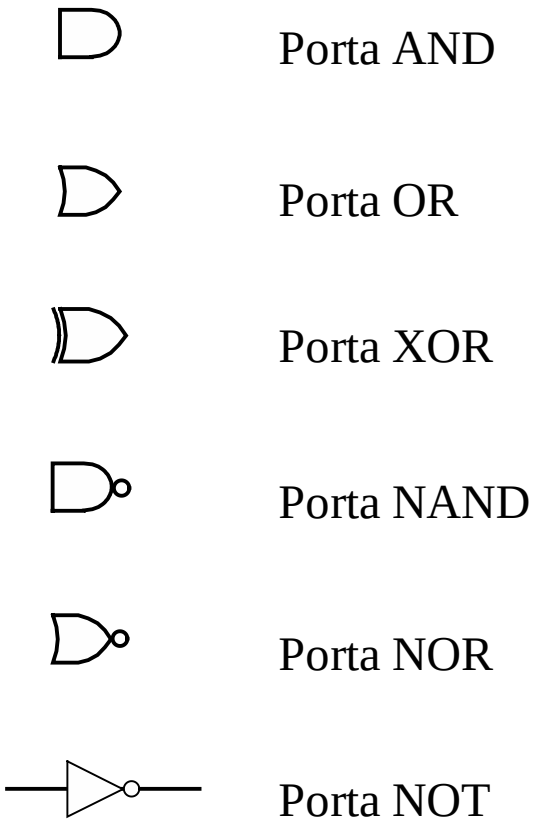

Figura 3.4: Tipi di porte logiche.

Operatore	Significato	Tabella di veritá
$\wedge$	E	Tabella 3.4
$\vee$	O	Tabella 3.5
$\neg$	NON	Tabella 3.10
$\oplus$	O...OPPURE	Tabella 3.8
$\rightarrow$	IMPLICA	Tabella 3.11
$\leftrightarrow$	COIMPLICA (EQUIVALE LOGICAMENTE)	Tabella 3.12

Tabella 3.9: I piú comuni operatori logici del calcolo proposizionale

A	$\neg$ A
$\mathbb{T}$	$\mathbb{F}$
$\mathbb{F}$	$\mathbb{T}$

Tabella 3.10: Tabella di veritá del connettivo $\neg$.

3.6 Richiami di logica: il calcolo proposizionale

I sistemi logici noti come logiche di ordine 0 sono fondati sulla struttura priva di assiomi propri, e dotata perció di soli postulati logici chiamata calcolo proposizionale . I *connettivi logici* delle teorie logiche, ed in specie del calcolo proposizionale sono molteplici. In Tabella 3.9 elenchiamo quelli piú usati comunemente, affiancati dal loro significato concettuale. Per ciascuno di questi la terza colonna della Tabella 3.9 contiene il riferimento alla tabella dove appare la forma della operazione definita dal connettivo. Tale forma appariá come tabella di veritá ovvero come una tabella a doppia entrata in cui leggeremo $\mathbb{T}$ per vero e $\mathbb{F}$ per falso. Le tabelle per gli operatori di congiunzione logica ($\wedge$), di disgiunzione logica ($\vee$), di negazione ($\neg$) e di disgiunzione esclusiva ($\oplus$) appaiono in tabelle della sezione 3.5

Finalmente, in Tabella 3.13 riportiamo un sunto delle espressioni per $\oplus$, $\rightarrow$ e $\leftrightarrow$ che usano i soli connettivi $\wedge$, $\vee$ e $\neg$.

A	B	A $\rightarrow$ B
$\mathbb{F}$	$\mathbb{F}$	$\mathbb{T}$
$\mathbb{F}$	$\mathbb{T}$	$\mathbb{T}$
$\mathbb{T}$	$\mathbb{F}$	$\mathbb{F}$
$\mathbb{T}$	$\mathbb{T}$	$\mathbb{T}$

Tabella 3.11: Tabella di veritá del connettivo $\rightarrow$.

A	B	A $\leftrightarrow$ B
$\mathbb{F}$	$\mathbb{F}$	$\mathbb{T}$
$\mathbb{F}$	$\mathbb{T}$	$\mathbb{F}$
$\mathbb{T}$	$\mathbb{F}$	$\mathbb{F}$
$\mathbb{T}$	$\mathbb{T}$	$\mathbb{T}$

Tabella 3.12: Tabella di veritá del connettivo $\leftrightarrow$.

Connettivo	Espressione in $\wedge$, $\vee$ e $\neg$
$A \oplus B$	$(A \wedge \neg B) \vee (\neg A \wedge B) =$ $(A \vee B) \wedge \neg(A \wedge B)$
$A \rightarrow B$	$A \vee \neg B$
$A \leftrightarrow B$	$(A \wedge B) \vee \neg(A \vee B) =$ $(A \vee \neg B) \wedge (\neg A \vee B)$

Tabella 3.13: Tabella delle espressioni dei connettivi $\oplus$, $\rightarrow$ e $\leftrightarrow$ mediante $\wedge$, $\vee$ e $\neg$.

3.7 Circuiti digitali e formule logiche proposizionali

Nei precedenti due paragrafi abbiamo introdotto tre connettivi non elementari, $\oplus$, $\rightarrow$ e $\leftrightarrow$. Nel paragrafo 3.4 abbiamo introdotto alcuni elementi di calcolo aritmetico in base 2. Questo paragrafo é dedicato alla costruzione di un circuito digitale che calcoli la somma in base 2.

Prodromicamente costruiremo i circuiti per $\oplus$, $\rightarrow$ e $\leftrightarrow$. Per tutti useremo come base di costruzione le espressioni logiche della Tabella 3.13. Come criterio guida, in caso di ambiguitá come avviene per $\leftrightarrow$, la scelta si fonderá sul criterio di evitare il piú possibile surriscaldamento, il che é espresso per i filtri nel paragrafo 3.5, come minimizzazione delle porte NOT. Per il caso di $\oplus$ la scelta ragionevole cade sull'espressione $A \vee B \wedge \neg(A \wedge B)$, mentre per il caso di $\leftrightarrow$ ricade su $(A \wedge B) \vee \neg(A \vee B)$.

In Figura 3.5 compare la forma del circuito per l'implicazione logica; in Figura 3.6 compare la forma del circuito per la disgiunzione esclusiva; in Figura 3.7 compare la forma del circuito per la coimplicazione.

Se consideriamo la cifra piú significativa della somma binaria di due numeri ad una cifra, vedremo che essa é esattamente una congiunzione logica (essendo infatti pari ad 1 solo per entrambi gli addendi pari ad 1). Viceversa, la cifra meno significativa risulta 0 se gli addendi sono o entrambi a 0 od entrambi ad 1, mentre quando degli addendi uno é 1 e l'altro é 0. Tale

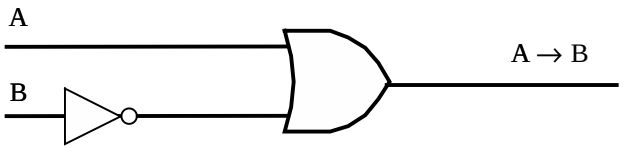

Figura 3.5: Circuito logico per l'implicazione.

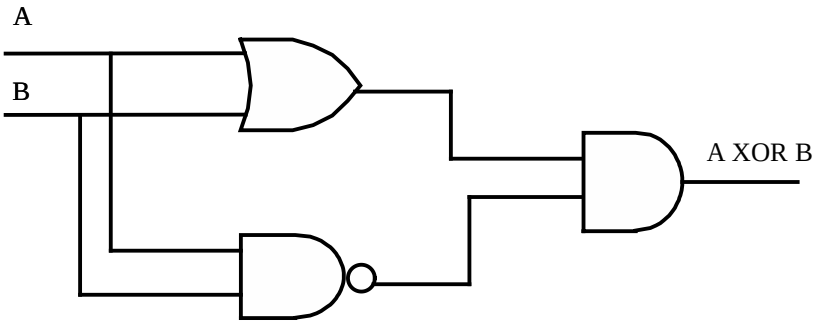

Figura 3.6: Circuito logico per la disgiunzione esclusiva.

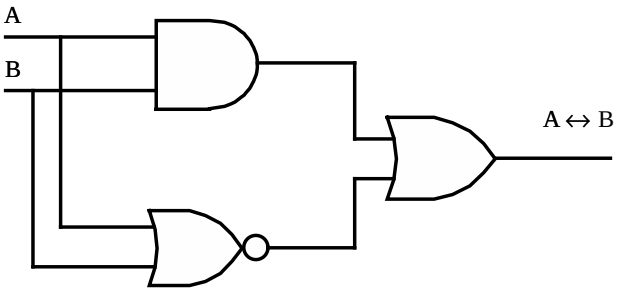

Figura 3.7: Circuito logico per la coimplicazione.

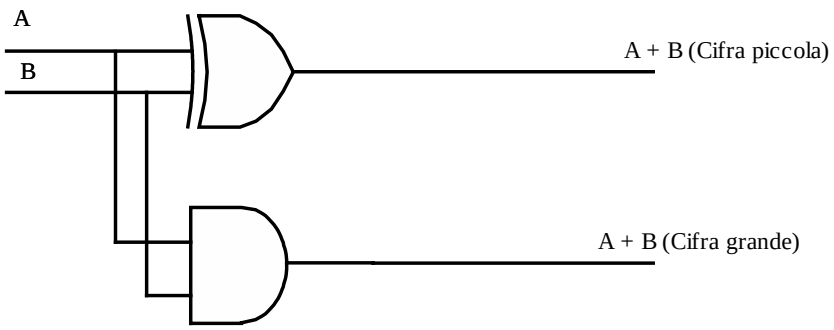

Figura 3.8: Circuito logico per la somma binaria.

operatore significa perció una disgiunzione esclusiva. Il circuito di Figura 3.8, rappresenta la somma di due numeri ad una cifra.

Capitolo 4

Elementi di teoria degli automi e della computazione

4.1 Introduzione

Il presente capitolo illustra, in modo introduttivo, i fondamenti teorici della teoria della computazione. Intordurremo tre aspetti chiave:

- La nozione di *macchina a stati* su cui poggia la teoria degli automi;
- La nozione di *funzione parziale ricorsiva* su cui poggia la teoria della calcolabilitá;
- Il concetto di *complessitá del calcolo* su cui poggia la teoria della complessitá o delle funzioni praticamente calcolabili.

Non avendo lo spazio per discutere in modo esaustivo le problematiche suddette, dopo averne enunciato le definizioni di base e le principali proprietà, discuteremo, in linea generale, di uno dei capisaldi concettuali della teoria della computazione, l'equivalenza di formalismi di calolo, e ci avvarremo di un esempio, la corrispondenza tra grammatiche regolari e automi a stati finiti, per chiarirne l'impatto.

4.2 Macchine a stati e concetto di calcolo

La nozione di macchina a stati deriva dall'idea, del tutto generale, che il calcolo consista nell'effettuazione di operazioni che dipendono dalla sequenza precedente e da informazioni fornite dall'esterno. Un sistema di tale tipo è in

effetti dotato di analoghe capacitá (di calcolo) rispetto ad un sistema prodotto dalla moderna ingegneria meccanica, od elettronica, dal telaio Jacquard alle macchine per il supercalcolo.

È importante che chi si avvicina alla disciplina informatica per la prima volta non abbia dubbi sul significato della nozione di "potenza di calcolo". Quando diciamo che una macchina a stati astratta come la macchina di Turing, o le Grammatiche di tipo 0, o il lambda calcolo sono, se vale la tesi di Church-Rosser, il formalismo di calcolo più potente che si può concepire intendiamo che se un problema non può essere risolto da una macchina astratta di tale tipo allora non esistono macchine in grado di risolverlo, nè se ne potranno concepire.

4.3 Gerarchia delle macchine a stati

La definizione formale di macchina a stati puó essere approfondita in molti modi. La scelta che effettuiamo é quella di dare un modello informale per le macchine di Turing e per gli Automi a pila ed introdurre formalmente gli Automi a stati finiti. Per gli scopi che ci prefiggiamo é nozione sufficiente quest'ultima.

Definizione 1 (Automa a stati finiti)
Una quintupla

$$\langle \Sigma, S, s_0, F, \delta(\cdot,\cdot) \rangle$$

dove:

- Σ *é un insieme finito e nonvuoto detto* alfabeto di input*;*
- S *é l'insieme degli* Statidell'automa*;*
- $s_0 \in \S$ *é lo* stato iniziale*;*
- $F \subseteq S$ *é l'insieme degli stati finali;*
- $\delta(\cdot,\cdot)$ *é una funzione definita su* $\Sigma \times S$ *a valori in* S *detta funzione di transizione.*

Parliamo di *linguaggio riconosciuto* da un automa riferendo l'insieme di tutte e sole le stringhe per cui l'automa passa susseguentemente dallo stato iniziale ad uno degli stati finali.

La funzione definita per estensione delle transizioni da stato a stato, corrispondente a stringhe dell'alfabeto di input. La definizione risulta pertanto:

$$\delta^*(s, \alpha) = s'$$

funzione di transizione estesa, con s ed s' stati dell'automa, e $\alpha \in T \supseteq \Sigma^*$ stringa, vale se e solo se:

$$\exists s = s_1, s_2, \ldots, s_n = s'[\alpha = a_1 a_2 \ldots a_n \rightarrow \forall i[1 \leq i \leq n - 1 \delta(s_i, a_i) = s_{i+1}$$

Negli assiomi di automa a stati finiti non é richiesto che la funzione di transizione sia definita per ogni simbolo e per ogni stato. Per ottenere una definizione di funzione di transizione estesa anche per i casi in cui la stringa contenga lettere che lasciano indefinita la funzione di transizione per lo stato in cui vengono lette occorre dire che cosa vale astrattamente la funzione di transizione in quei casi. Abbiamo tre scelte:

- *Interpretazione restrittiva*: l'automa smette di funzionare correttamente ogni volta che legge un simbolo di input per il quale la funzione di transizione non é definita nello stato in cui l'automa si trova;
- *Interpretazione estensiva*: come nella restrittiva, tranne che per gli stati finali dove l'automa persiste dopo la lettura di qualsiasi simbolo di input anche se la funzione di transizione non fosse definita;
- Interpretazione totalmente estensiva: l'automa persiste in uno stato ogni volta che legge un simbolo di input per cui la funzione di transizione non é definita in quello stato.

Per ottenere un comportamento corretto nelle interpretazioni restrittiva ed estensiva é necessario introdurre un nuovo simbolo di stato noto con il nome di *violazione* o *stato di blocco*, generalmente indicato con $\perp$. Tale stato, non finale, viene aggiunto agli stati e da ogni stato (non finale per l'interpretazione estensiva) l'automa transita a $\perp$ se legge simboli per cui non é prevista transizione. La figura 4.1 mostra un automa a stati finiti e la loro modifica in presenza dello stato di blocco.

Precisiamo che gli automi cosí definiti sono *deterministici*. Se estendiamo la definizione in modo che invece di avere una funzione di transizione da coppie stato-simbolo in stati abbiamo definito la funzione con codominio nell'insieme delle parti dell'insieme degli stati, gli automi risultano nondeterminitici. I linguaggi riconosciuti dagli automi determintici sono peró gli stessi di quelli riconosciuti da quelli deterministici.

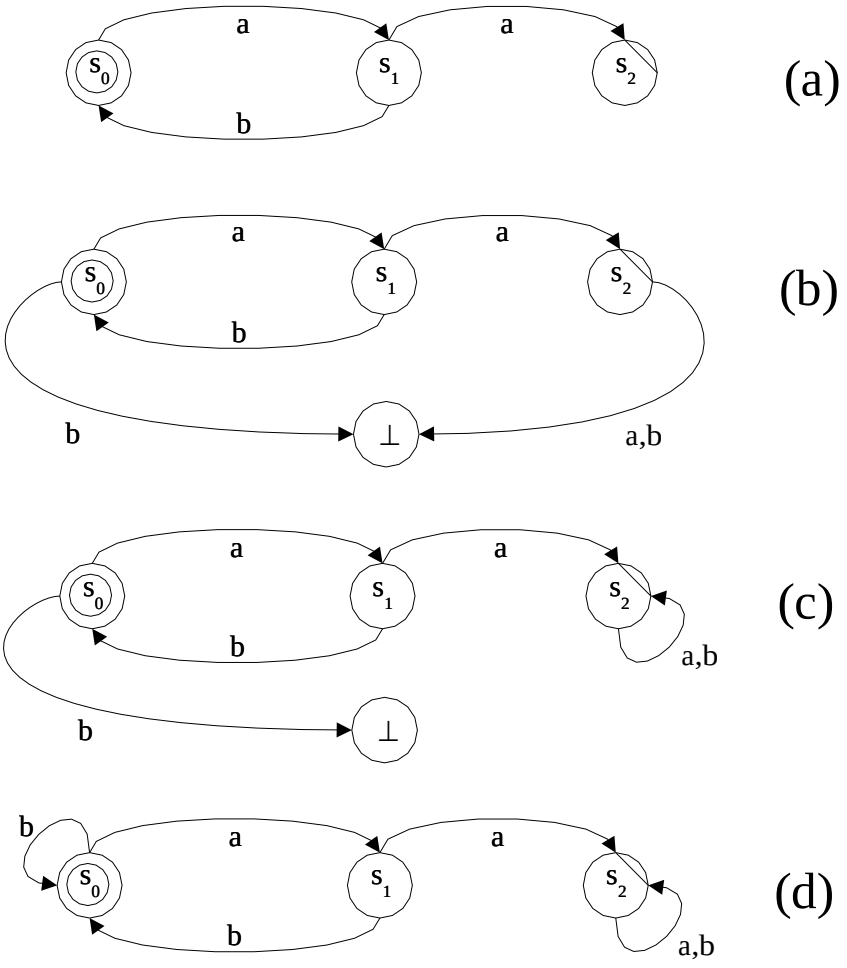

Figura 4.1: Un automa a stati finiti (a) e la forma che si ottiene con l'interpretazione restrittiva (b), estensiva (c) e totalmente estensiva (d). Notate lo stato di blocco aggiunto per (b) e (c).

Macchine a stati	***Grammatiche***
Automi a stati finiti	Regolari
Automi a pila deterministici	Liberi dal contesto determinsitici
Automi a pila	Liberi dal contesto
Macchine di Turing a nastro limitato	Sensibili al contesto
Macchine di Turing	Di tipo 0

Tabella 4.1: La gerarchia delle macchine a stati definita secondo le classi di riconoscimento.

Nel prossimo paragrafo introdurremo il formalismo delle grammatiche generative. Esso definisce una gerarchia di linguaggi dai piú semplici (regolari) fino a quelli di tipo 0, con due classi principali (liberi dal contesto e sensibili al contesto. I linguaggi regolari sono riconosciuti dagli automi a stati finiti.

Dato che conosciamo l'esistenza di linguaggi piú complessi dei regolari (liberi dal contesto, sensibili al contesto, di tipo 0) é lecito supporre che esista una gerchia delle macchine a stati che rifletta tale strutturazione. In effetti, una prima estensione agli automi a stati finiti si ottiene definendo gli *automi a pila*, che sono fatti come gli automi a stati finiti, tranne che per la disponibilitá di una memoria accessibile secondo le modalitá della pila (ovvero inserendo un elemento all'ultimo posto ed estraendo l'ultimo elemento inserito) dove la macchina puó scrivere e da cui legge l'input. La presenza di una memoria rende gli automi di questo tipo piú potenti di quelli a stati finiti. Si dimostra, in particolare, che gli automi a pila deterministici riconoscono linguaggi liberi dal contesto deterministici, mentre gli automi a pila nondeterministici riconoscono tutti i linguaggi liberi dal contesto.

Il limite degli automi a pila sta essenzialmente nell'unidirezionalitá della memoria. Se costruiamo una macchina che legge e scrive in due direzioni su di un nastro otteniamo la *macchina di Turing*. Se il nastro é di lunghezza finita la macchina verrá detta *a nastro limitato*. Si dimostra che le macchine a nastro limitato riconoscono i linguaggi sensibili al contesto, mentre quelle generali riconoscono i linguaggi di tipo 0. Le corrispondenze tra macchine a stati e grammatiche sono riassunte in Tabella 4.1.

4.4 Grammatiche generative

Il concetto di grammatica generativa venne sviluppato nell'ambito della linguistica come modello matematico della produzione sintatticamente corret-

ta. Al di là dei natali il modello ebbe uno straordinario successo in matematica ed in informatica teorica nella rappresentazione dei linguaggi formali, che sono stati adottati come modello del calcolo, e dimostrati infatti equipotenti con le macchine di Turing, coerentemente con la tesi di Church-Rosser.

Prima di addentrarci nella definizione di grammatica richiamiamo i concetti elementari su cui tale nozione è costruita.

Definizione 2 *Un alfabeto è un insieme finito e non vuoto i cui elementi vengono detti simboli.*

Definizione 3 *L'insieme di tutte le successioni finite (concatenazioni) di simboli di un alfabeto A, è detto* chiusura *di A ed indicato da A^*. L'insieme di tutte le stringhe nonvuote di simboli su un alfabeto A è indicato da A^+, mentre la stringa vuota viene indicata da ϵ.*

Definizione 4 *Ogni sottoinsieme L della chiusura di un alfabeto A è detto linguaggio su A, $L \subseteq A^*$.*

Definizione 5 *Una grammatica formale G è una quadrupla $\langle T, NT, \nu, P \rangle$, dove:*

- *T è un alfabeto, detto* alfabeto terminale*;*
- *NT è un alfabeto, detto* alfabeto nonterminale*, disgiunto da T;*
- *ν è un elemento di NT detto* simbolo distinto*;*
- *P è un insieme di coppie (α, β) rappresentanti* regole di produzione *di stringhe, dette rispettivamente* produttore *e* prodotto*, tali che $\alpha, \beta \in (T \cup NT)^*$, contenente almeno una coppia il cui produttore è ν. Una regola di produzione verrá indicata da $\alpha \rightarrow \beta$.*

Data una grammatica formale G, ciò di cui vogliamo occuparci è la nozione di linguaggio generato da G. Per fare questo occorre definire il concetto di *derivazione*, mediante una grammatica G.

Definizione 6 *Data una grammatica G e due stringhe x ed y, $x, y \in (T \cup NT)^*$, diremo che x deriva direttamente y mediante G, indicato da $x \Rightarrow y$, se e solo se $x = as_1b$ ed $y = as_2b$, con a e b entrambi in $(T \cup NT)^*$, ed esiste una regola in G tale che $s_1 \rightarrow s_2$. Diremo viceversa che x deriva indirettamente y se e solo se x deriva z che deriva direttamente y. La derivazione indiretta verrá indicata da $x \overset{*}{\Rightarrow} y$*

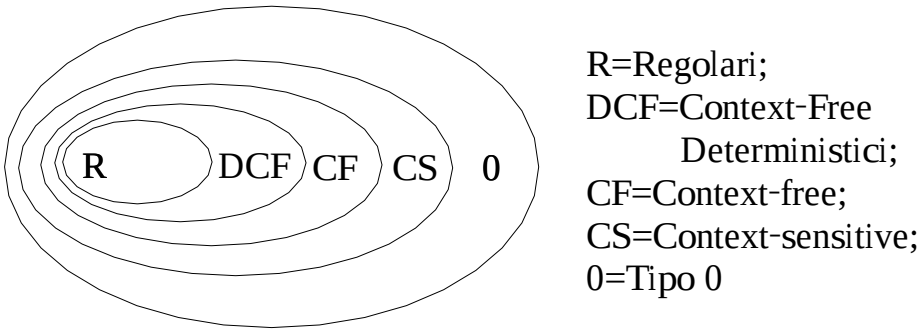

Figura 4.2: Le cinque classi di grammatiche generative e le loro relazioni di inclusione rappresentate graficamente.

Definizione 7 *Chiamiamo* linguaggio generato da una grammatica G *l'insieme delle stringhe terminali che possono essere derivate dal simbolo distinto. Formalmente* $L(G) = \{\sigma \in T^* | \nu \overset{*}{\Rightarrow} \sigma\}$

La forma delle regole di produzione determina una gerarchia delle grammatiche in cinque livelli: *regolari*, *libere dal contesto deterministiche*, *libere dal contesto*, *sensibili al contesto*, *di tipo 0.* Il grafico qui sotto illustra le relazioni di inclusione tra le classi che determinano la struttura gerarchica.

In particolare, le quattro classi su introdotto sono identificate dalle forme delle regole di produzione come nelle seguenti definizioni.

Definizione 8 *Una grammatica* $G = \langle T, NT, \nu, P \rangle$*, si dice regolare destra se e solo se ogni regola di produzione in* P *é della forma:*

$$A \to a$$

o della forma

$$A \to aB$$

dove A, B *sono simboli nonterminali e* a *é un simbolo terminale.*

Analogamente, una grammatica $G = \langle T, NT, \nu, P \rangle$*, si dice regolare sinistra se e solo se ogni regola di produzione in* P *é della forma:*

$$A \to a$$

o della forma

$$A \to Ba$$

dove A, B sono simboli nonterminali e a é un simbolo terminale.

L'insieme delle grammatiche regolari destre verrà indicato con RR mentre quello delle grammatiche regolari sinistre verrà indicato con LR. L'insieme dei linguaggi generati con grammatiche di una classe C verrà indicato con $\mathcal{L}(C)$.

Teorema 1 *La classe $\mathcal{L}(RR)$, dei linguaggi regolari destri, coincide con la classe $\mathcal{L}(LR)$ dei linguaggi regolari sinistri.*

Definizione 9 *Dato un alfabeto Σ, ed una stringa σ su Σ, chiamiamo* espressione regolare elementare *in Σ, con base σ un insieme tra*

1. *il singoletto contenente σ;*
2. *l'insieme di tutte le concatenazioni di σ, inclusa la concatenazione vuota (σ^*).*

Data una successione di stringhe $\sigma_1, \sigma_2, \ldots, \sigma_n$, chiameremo espressione regolare semplice *la concatenazione secondo l'ordine della successione delle espressioni regolari elementari in Σ con base σ_i in posizione i. L'unione di espressioni regolari semplici verrá detta* espressione regolare.

L'espressione regolare $\sigma\sigma^*$ verrá sinteticamente indicata con σ^+.È facile dimostrare i seguenti teoremi.

Teorema 2 *Ogni espressione regolare è un linguaggio regolare*

Dimostrazione Data l'espressione regolare elementare σ, con $\sigma = s_1 s_2 \ldots s_k$, le regole di produzione:

$$\begin{aligned} A_0 &\to s_1 A_1 \\ A_1 &\to s_2 A_2 \\ &\ldots \\ A_{k-1} &\to s_k \end{aligned}$$

generano la stringa σ. L'espressione regolare elementare σ^* si genera a partire dalle regole

$$\begin{aligned} A_0 &\to \epsilon \\ A_0 &\to s_1 A_1 \\ A_1 &\to s_2 A_2 \\ &\dots \\ A_{k-1} &\to s_k \\ A_{k-1} &\to s_k A_0 \end{aligned}$$

Aggiungendo opportunamente una sequenza di nonterminali come nell'esempio suddetto si ottiene il corretto insieme di regole di produzione per ogni espressione regolare.

■

Teorema 3 *Ogni linguaggio regolare è un'espressione regolare.*

4.5 Formalismi di calcolo e loro equivalenza: la tesi di Church-Rosser

La comune sensibilitá tra informatici e matematici a riguardo del problema di che cosa possa essere calcolato condusse alla definizione informale di calcolabilitá nota con il nome di calcolabilitá alla Hilbert

Definizione 10 *Sia f una endofunzione sui naturali, ovvero*

$$f : \mathbb{N} \to \mathbb{N}$$

allora diremo che f é calcolabile sse *esiste un* algoritmo *(ovvero una macchina a stati) che per qualsiasi $n \in \mathbb{N}$, calcola $f(n)$.*

Le funzioni calcolabili vengono anche chiamate *ricorsive*. Ovviamente, valendo la definizione, le funzioni suddette sono parziali e perció il calcolo é definito dove é definita la funzione, ovvero sono parzialmente computabili.

É terminologia intercambiabile con calcolabile: ricorsivo, computabile e decidibile.

Una definizione analoga, piú diretta rispetto ai problemi dell'informatica si puó ottenere valutando l'esistenza di modi di ottenere il successore di un numero di $\mathbb{N}$ in un insieme.

Definizione 11 *Sia $S \subseteq \mathbb{N}$ un insieme di numeri naturali.*

S é ricorsivamente enumerabile*,* sse *esiste un modo per ottenere il successore di n in S[1] per ogni $n \in S$ in tempo finito.*

S é ricorsivo sse *esiste un modo per decidere se n appartiene ad S per ogni $n \in \mathbb{N}$.*

Ogni insieme ricorsivo é anche ricorsivamente enumerabile, infatti per calcolare il successore di un dato n in S basta decidere se $n + i$ appartiene ad S per ogni $i > 0$.

Tutte le definizioni suddette sono astratte, cioé indipendenti dalla specifica macchina a stati che esegue il compito richiesto (calcolare una funzione, decidere l'appartenenza ad un insieme di naturali). Una naturale distinzione che desideriamo fare é tra ció che si puó calcolare con una macchina di Turing e ció che si puó calcolare con altri formalismi ipoteticamente piú potenti. Ció che si calcola con la macchina di Turing cade sotto il nome di *Turing-computabile*. La tesi di Church viene cosí enunciata:

Tesi di Church
Ogni funzione che sia computabile é Turing-computabile.

Se la tesi di Church fosse vera, le Macchine di Turing non sarebbero solo il piú potente formalismo di calcolo conosciuto, ma il piú potente possibile. Finora la tesi di Church non é mai stata smentita e ci sono varie prove che indicano che con tutta probabilitá essa é vera, anche se non si é ancora trovato modo di dimostrarla formalmente.

4.6 Le grammatiche regolari e gli automi a stati finiti

In questo paragrafo mostriamo tre tecniche di traduzione che servono come esempio di corrispondenza tra classi delle gerarchie di macchine e di grammatiche precedentemente descritte:

- Una tecnica di traduzione che conduce da un automa a stati finiti ad una grammatica regolare;
- Una tecnica che trasforma un automa nondeterministico in uno deterministico;

[1] In un insieme di numeri naturali S si dice successore di un numero $n \in S$ il piú piccolo $m \in S$ che soddisfa $n < m$.

- Una tecnica che traduce da grammatiche regolari in automi a stati finiti nondeterministici.

Dagli automi alle grammatiche Sia $\mathcal{A}$ l'automa a stati finiti $\langle \Sigma, S, s_0, F, \delta(\cdot,\cdot)\rangle$. Definiamo la grammatica $\mathcal{G} = \langle T, NT, \nu, P\rangle$ nel modo seguente:

- Per ogni stato di $\mathcal{A}$ introduciamo un simbolo nonterminale per $\mathcal{G}$ in NT;
- Assegnamo s_0 a ν;
- Per ogni transizione $\delta(s,a) = s'$ introduciamo la regola di $\mathcal{G}$ in P, $s \rightarrow as'$;
- Per ogni transizione $\delta(s,a) = s'$ con s' finale, introduciamo la regola di $\mathcal{G}$ in P, $s \rightarrow a$.

É facile dimostrare che il linguaggio generato da $\mathcal{G}$ é esattamente quello riconosciuto da $\mathcal{A}$.

Da automi nondeterministici ad automi deterministici Sia A un automa nondeterministico. Si definisca un nuovo automa A', deterministico, ottenuto come segue:

- Per ogni sottoinsieme nonvuoto dell'insieme degli stati di A, esiste uno stato in A';
- Il singoletto dello stato iniziale di A é lo stato iniziale di A';
- Ogni sottoinsieme di stati di A che contenga uno stato finale é finale in A';
- Per ogni transizione definita in A su di uno stato $s$per un certo input x, $\delta(s,x) = D \subseteq S$ dove S é l'insieme degli stati di A, l'automa A' transita su D dal singoletto di s leggendo x;
- Per ogni sottoinsieme di stati D di A, l'automa A' transita sull'unione dei sottoinsiemi di stati corrispondenti alle transizioni dei singoletti in D.

Da grammatiche ad automi nondeterministici Sia $\mathcal{G} = \langle T, NT, \nu, P \rangle$ una grammatica regolare. Definiamo l'automa $\langle \Sigma, S, s_0, F, \delta(\cdot, \cdot) \rangle$ che riconosce il linguaggio generato da $\mathcal{G}$ nel modo seguente:

- Per ogni simbolo nonterminale di $\mathcal{G}$, l'automa possiede uno stato;
- Lo stato corrispondente al simbolo distinto é iniziale;
- Uno stato speciale F, finale, esiste nell'automa;
- Per ogni regola di produzione $A \rightarrow bB$ gli stati A e B sono connessi da una transizione per lettura di b;
- Ogni regola di produzione $A \rightarrow b$ si traduce in una transizione da A a F per la lettura di b.

4.7 Complessitá del calcolo

Il presente paragrafo descrive informalmente la nozione di complessitá del calcolo. Per prima cosa definiamo la nozione in modo strutturato. Per calcolare la complessitá del calcolo occorre avere a disposizione la nozione di *passo di calcolo*. Nell'esecuzione di un programma un passo puó essere:

- Una generica istruzione;
- Un assegnamento;
- La verifica di una condizione.

In ciascuno dei suddetti casi, il calcolo ha un *costo computazionale* definibile come la funzione che associa alla lunghezza dell'input il numero di passi che devono essere eseguiti da quel programma per le istanze di quella lunghezza. Avremo in specie tre casi:

- Il caso migliore, cui associamo la funzione che descive il numero minimo di passi per una definita lunghezza;
- Il caso medio, che ovviamente definisce la funzione che associa il numero medio di passi ad una lunghezza;
- Il caso peggiore, che associa il numero massimo di passi ad una lunghezza.

Due programmi potrebbero implementare metodi di risoluzione differenti, oppure lo stesso metodo (ad esempio in piú linguaggi) con differenti tecniche. Se due programmi implementano lo stesso metodo diremo che corrispondono allo stesso algoritmo.

Definizione 12 *Chiamiamo complessitá computazionale di un algoritmo, il costo computazionale della migliore implementazione possibile di quell'algoritmo, ovvero il costo computazionale del programma che meglio realizzerebbe l'algoritmo stesso una volta implementato.*

Definizione 13 *Chiamiamo complessitá computazionale di un problema, il costo computazionale del miglior algoritmo risolutore possibile.*

4.8 Problemi teoricamente e problemi praticamente risolubili

Per definire la differenza tra problemi che ha senso tentare di risolvere e problemi che per la loto complessitá suggeriscono implementazioni incomplete, occorre rilevare che i problemi la cui natura é intrinsecamente troppo complessa per essere risolti nella pratica, sono quelli esponenziali. All'opposto, i problemi che possiamo ragionevolmente risolvere in pratica sono quelli il cui costo computazionale risulti polinomiale.

Una ulteriore distinzione assai utile nella pratica si ha definendo la nozione di riduzione polinomiale. Un problema p si dice polinomialmente riducibile ad un problema q *sse* esiste un metodo per tradurre ogni istanza di p in una istanza di q in tempo polinomiale. Chiaramente, se q é polinomialmente risolubile, lo sará anche q.

Infine, i problemi si distinguono in risolubili in tempi polinomiali su macchine deterministiche e nel superinsieme costituito da tutti quei problemi che si risolvono in tempi polinomiali su macchine noindeterministiche. I primi venogono in genere indicati con il nome di classe P, mentre i secondi vengono indicati con il nome di classe NP. Chiaramente $P \subseteq NP$. Il problema di stabilire se $P \subset NP$ ovvero se $P = NP$ é aperto.

Sia $\mathcal{A}$ una classe di problemi. Un generico problema p si dice $\mathcal{A}$-hard *sse* esiste un metodo per ridurre tutti i problemi di $\mathcal{A}$ a p. Se p é $\mathcal{A}$-hard, ed é in $\mathcal{A}$, allora é NP-completo. I problemi piú interessanti da questo punto vista, sono quelli NP-completi.

Infatti se uno specifico problema X risulta essere NP-completo, e troviamo per questo una tecnica di risoluzione polinomiale su macchine deterministiche, ogni problema di NP risulta polinomialmente risolubile. Basterá

infatti tradurre un qualsiasi problema di NP (in tempo polinomiale, per definizione) nel problema X, e poi risolvere X (ancora, in tempo polinomiale, per definizione).

Capitolo 5

Introduzione ai sistemi operativi

5.1 Introduzione

In questo capitolo illustreremo le principali caratteristiche tecniche e strutturali dei moderni sistemi operativi. In particolare, nella sezione 5.2 discuteremo della nozione concettuale di sistema operativo, nella sua semiformale definizione ingegneristica; nella sezione 5.3 diremo quali tecniche debbano essere usate per risolvere il principale problema del livello software piú vicino alla macchine dei sistemi operativi, noto con il nome di Job-Shop Scheduling Problem; nella sezione 5.4 descriveremo la struttura dei sistemi operativi dal punto di vista concettuale introducendo quella che é generalmente nota con il nome di struttura a cipolla (per la forma a strati successivi); nella sezione 5.5 descriveremo le componenti logiche di un ambiente operativo, ed infine nella sezione 5.6 introdurremo la nozione di autorizzazione e di ruolo in un sistema operativo.

5.2 Nozione di sistema operativo

La nozione di sistema operativo si ottiene modificando la nozione generale di sistema dell'ingegneria, attribuendo un ruolo al sistema, che deriva dalla concezione di operativitá dell'architettura di van Neumann. In effetti esso si ottiene a partire dalla nozione di sistema introducendo i concetti di :

- regolazione diretta ed inversa;
- retroazione;

- agente-utente.

Per illustrare il concetto di sistema occorre precisare le nozioni su cui esso poggia. Esse sono la nozione di *componente*, *struttura* e *comportamento*.

Chiamiamo componente di un sistema un oggetto interdipendente, la cui esistenza e funzione siano direttamente correlate con l'esistenza e la funzione del sistema cui si relaziona. Cosí, ad esempio, in un'automobile, l'impianto elettrico é una componente, ed in questo caso l'interdipendenza si manifesta nell'assenza di funzione della componente isolata (non si puó concepire l'impianto elettrico come a sé stante) e nella mancanza di funzione (quella svolta dall'impianto elettrico) nel sistema. Ovviamente la nozione di componente viene espressa in astratto, ed é perció concepibile un sistema che ha componenti che a loro volta sono sistemi, ed inoltre possono esistere tipi diversi di interdipendenza, in specie, l'esistenza e funzione della componente possono dipendere singolarmente dall'esistenza e dalla funzione del sistema e viceversa. Sinteticamente:

- Ogni sistema ha almeno una componente;
- Ogni componente é interdipendente dal sistema in uno dei modi seguenti:
  - L'esistenza della componente dipende dall'esistenza del sistema (come nel caso della pareti di un carburatore, che non esistono se non esiste il carburatore);
  - La funzione della componente dipende dalla relazione con il sistema (come nel succitato esempio dell'impianto elettrico per un'automobile);
  - La funzione del sistema dipende dalla funzione della componente (come nel caso dell'automobile per, ad esempio, il motore, o ancora lo stesso impianto elettrico).

Chiamiamo struttura di un sistema il complesso delle relazioni esistenti tra le componenti.

Chiamiamo infine comportamento di un sistema il complesso delle relazioni esistenti tra il sistema e l'universo in cui il sistema é immerso. Perché un complesso di oggetti relazionati possa essere considerato un sistema occorre che sussista la suddetta interdipendenza tra componenti e sistema, e che esista analoga interdipendenza tra comportamento delle componenti e comportamento del sistema stesso.

Dato un sistema S, una componente C di S si dice *regolare* direttamente S, se e solo se alcuni comportamenti di C deterministicamente vincolano comportamenti di S. Si dice viceversa che C regola inversamente S se e solo se alcuni comportamenti di C condizionano S, deterministicamente, a *non* avere certi comportamenti. Si chiama retroazione il fatto che certi comportamenti di un sistema S condizionino direttamente o inversamente comportamenti delle sue componenti. Una retroazione verrá detta positiva se le componenti retroregolate sono regolatori del sistema stesso, essendo pertanto il comportamento del sistema dipendente non solo da ció che accade all'esterno, ma anche da ció che il sistema ha fatto fino a quel momento.

Se un sistema é dotato della capacitá di autoregolazione si dirá autonomo. Chiameremo stato di un sistema una descrizione delle condizioni in cui si trovano le sue componenti e delle relazioni che intercorrono tra le componenti stesse. Se una componente regolatore é dotato delle capacitá di scambiare informazioni dal di fuori del sistema, tale componente si dirá una interfaccia. Qualsiasi sistema che possa fornire informazioni ad un altro sistema attraverso una coppia di interfaccie verrá definito una componente esterna. Se un sistema é dotato di una componente esterna che sia regolatore diretto o inverso, diremo che tale componente é un agente del sistema. Se gli effetti di retroazione del sistema agiscono sulla medesima componente esterna, allora diremo che quell'agente é anche un utente del sistema, e le azioni di retroazione verso tale componente esterna si diranno servizi del sistema.

Sia data una architettura di Van Neumann X ed un sistema Y dotato di capacitá di regolazione verso X. Se tutte le *operazioni* che possono essere compiute da X sono sotto il controllo delle regolazioni di Y diremo che Y é un *sistema operativo*per X.

Le nozioni chiave del sistema operativo, che verranno poi esplicitamente descritte nei prossimi paragrafi sono tre:

- *Archivio* o *File* che é una sequenza di dati organizzati secondo schemi dove informazioni che devono essere conservate in modo persistente risiedono, tipicamente in un disco o in una differente unitá di I/O;

- *Programma* o *Applicazione* o *Comando* é un file che puó essere eseguito;

- *Processo* é un programma in esecuzione.

5.3 Concorrenza di processi: schedulazione

La nozione piú rilevante di cui dobbiamo dare conto in un sistema operativo é quella di concorrenza. Per comprendere l'importanza di tale nozione occorre rilevare alcune caratteristiche tecnologiche dell'architettura di Van Neumann che ci dicono quale metodologia deve essere seguita al fine di risolvere il problema cosí come l'abbiamo enunciato.

In primo luogo, avendo definito astrattamente un programma in funzione della nozione di macchina, nel precedente capitolo, dobbiamo fornire la definizione di ció che effettivamente occorre in questo contesto. Un programma in esecuzione su una macchina fisica si chiamerá un processo.

In particolare, come risulta chiaramente dalla definizione strutturale, nell'architettura di Van Neumann, un solo programma puó essere eseguito alla volta, cioé la macchina di Van Neumann é una *macchina sequenziale*. Viceversa noi vorremmo, per evidenti ragioni pratiche, dotare un sistema operativo delle seguenti funzioni:

- *multi-user access*: cioé la possibilitá che piú utenti accedano simultaneamente alla macchina;
- *multi-task execution*: la possibilitá che piú programmi siano eseguiti simultaneamente.

Per la prima funzione occorre, evidentemente, dotare il sistema di piú basi di accesso dotate di input ed output. Il nome di queste basi d'accesso é, genericamente, *terminali*. Per la seconda, occorre dotare il sistema di uno *schema di condivisione delle risorse*, cioé di un insieme di regole che stabiliscano in modo non ambiguo a chi spetti eseguire calcoli, o accedere alla memoria, tra i processi che concorrono ad ottenere tali risorse. A causa della necessitá di tale schema parleremo, riferendoci alla componente di un sistema operativo che svolge tale funzione di *Resource manager*. La sottocomponente di tale sistema che in specie gestisce la distribuzione della risorsa temporale prende il nome di *Job Manager*. Oltre a tale componente si ritrovano il *Memory Manager* e la componente che gestisce la messa in coda dei lavori. Quest'ultima prende generalmente il nome di *Queue Manager*.

Il Resource Manager si avvale di una speciale componente, la cui funzione tenere sotto controllo l'accesso alle risorse e fornire le prioritá di accesso secondo lo schema. Questa sottocomponente chiamata il *Monitor*.

Uno schema di funzionamento del Resource Manager appare in Figura 5.1.

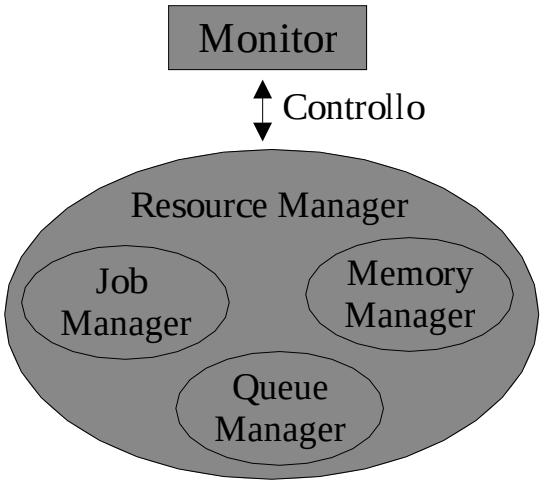

Figura 5.1: Schema del Resource Manager in un sistema operativo.

Storicamente le tecnologie con cui sono stati realizzati Job, Queue e Memory Manager sono diverse, anche in funzione dei differenti obiettivi delle componenti. In particolare, il Job Manager gestisce l'assegnazione del tempo alle risorse richiedenti, cioé ai programmi, e differenti strutture Hardware sono state concepite per migliorare l'efficienza di questa componente. Viceversa la gestione delle code é piú semplice, per cert versi avendo subito poco l'evoluzione dell'hardware. Infine la gestione della memoria ha seguito e segue regole diverse a seconda del tipo di memoria (centrale o di massa) e dell'esistenza di componenti hardware tecnologicamente innovative, quali in particolare, la memoria cache.

Lo schema storicamente utilizzato per l'accesso via job scheduler ad una unitá centrale di elaborazione é stato FIFO (First In First Out), ovvero la risorsa CPU veniva assegnata in ordine di richiesta. Questo metodo ha tre svantaggi:

- Processi con maggior urgenza di completamento sono trattati alla stregua di altri;
- Processi brevi vengono posti potenzialmente dopo processi lunghi;
- Se un processo deve accedere ad una risorsa esterna (disco, stampante), lascia comunque in attesa i processi successivi, pur possedendo la CPU senza utilizzarla.

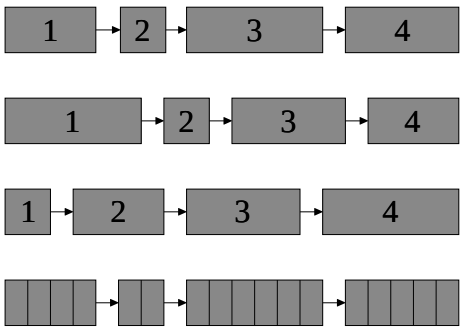

Figura 5.2: I quattro schemi principali di schedulazione in un sistema operativo. Nell'ultima riga l'esecuzione di ciascuna parte di job in coda avviene secondo l'ordine stabilito dal round-robin.

Per ovviare al primo degli inconvenienti suddetti vennero introdotti dei Job Scheduler che operavano a prioritá. Il secondo e terzo inconveniente peró rimasero irresoluti piú a lungo.

Il primo tentativo astrattamente utile di modifica alla strategia FIFO fu la invenzione della politica di schedulazione SJB (Shortest Job First). Questa politica consiste nello schedulare i Job in ordine crescente di dimensione. L'effetto di questa modifica é di garantire un tempo atteso di permanenza in coda che é proporzionale alla dimensione relativa di un job immesso. Per poter risolvere il terzo inconveniente occorre rivolgersi direttamente alle tecnologie hardware. Infatti per poter gestire questa modifica é necessario che ogni dispositivo esterno alla CPU) includa la memoria centrale sia dotato di un apposito dispositivo aggiuntivo, noto con il nome di DMA (Direct Memory Access). Se il sistema hardware su cui si appoggia un sistema operativo (oggi praticamente ovunque) é dotato di DMA allora possiamo immaginare un sistema di schedulazione che sottrae la CPU (competitivamente) ad una risorsa e l'assegna alla successiva in coda. Chiaramente diventa naturale sottrarre la CPU alla risorsa che la sta utilizzando ogni volta che il processo in esecuzione chiede di usare una DMA e comunque una volta trascorso un intervallo di tempo massimale noto con il nome di *Time-Slice*. La politica di schedulazione in questo caso diviene Round-Robin. Uno schema del funzionamento dei quattro sistemi di schedulazione suddetti (FIFO, FIFO con prioritá, SJB, Round-Robin) é mostrato in Figura 5.2.

Una descrizione formale del round-robin si ottiene con l'algoritmo di

```
Passo 1   Fino a quando ci sono job in coda che chiedono la risorsa CPU
Passo 2         Consegna la CPU al job corrente
Passo 3         Se Il job corrente non rilascia
                    la CPU per accedere a DMA prima
                    del time-slice allora
Passo 4                       Sottrai la CPU al job corrente
                              e consegnala al successivo Job in coda
```

Figura 5.3: Schema di un algoritmo di schedulazione round-robin.

Figura 5.3.

5.4 Struttura di un sistema operativo

I sistemi operativi sono generalmente descritti, nella corrente letteratura del settore come una struttura multilivello (gergalmente chiamata "a cipolla").

Il livello piú lontano dall'utente é quello della macchina fisica (Hardware). SI considera questo livello come facente parte dell'architettura di un sistema operativo per le dirette implicazioni architetturali dell'hardware nella strutturazione del Job Manager (ed in particolare dello scheduler) e del Memory Manager. Sono essenziali infatti tre componenti:

- Le DMA;
- Le memorie Cache della memoria centrale e dei dischi;
- Il Clock.

Il livello immediatamente successivo all'hardware consiste fondamentalmente nell'implementazione di Job Manager, Memory Manager, Queue Manager e del Monitor, cioé complessivamente del Resource Manager, nonché di alcune componenti collaterali quali specifici gestori di dispositivi esterni (Device Monitor) e di componenti per l'amministrazione di funzioni di accesso alla rete (Network Monitor).

Il terzo livello di un sistema operativo é costituito dai *Tools*, cioé programmi eseguibili che il sistema rende disponibili all'utente perché possano essere richiamate complesse funzioni ottenute attraverso il kernel. In questa parte del sistema risiedono la maggior parte delle fondamentali strutture che interfacciano il sistema con l'esterno. I comandi possono essere visti come

appartenenti ad uno dei seguenti gruppi, direttamente interconnessi con le componenti del kernel:

- Comandi di *Filesystem*;
- Comandi di *Monitoring*;
- Comandi di *Networking*;
- Comandi di *Comandi di gestione periferiche*;
- Comandi di attivazione di ambienti operativi testuali;
- Comandi di attivazione di ambienti operativi visuali.

I comandi di Filesystem permettono di gestire gli *archivi di informazione* che il sistema operativo permette di trattenere in modo persistente, nelle loro varie forme.

I comandi di Monitoring permettono di visualizzare e gestire lo stato di esecuzione dei processi e le loro relazioni.

I comandi di Networking interfacciano la macchina che interagisce con l'utente con altre macchine della *rete locale* o con macchine di *rete geografica*. Due macchine si dicono in rete locale se esiste una connessione diretta tra di loro o una singola macchina in grado di interfacciarle.

I comandi di Gestione periferiche permettono di effettuare operazioni standard di gestione di periferiche quali mouse, tastiere, video, stampanti, dischi.

I comandi di attivazione sia per ambienti operativi testuali che visuali permettono di mettere in esecuzione il livello piú vicino all'utente, dal punto di vista interno al sistema: il livello dell'ambiente operativo. L'ambiente operativo é una interfaccia utente che permette all'utilizzatore di un sistema di richiamare tools (in particolare comandi) ed eseguire applicazioni, spesso considerate il successivo ed ultimo livello di un sistema operativo.

In Figura 5.4 viene mostrata una descrizione grafica di quanto esposto nel presente paragrafo.

5.5 Componenti logiche di un ambiente operativo visuale

Ogni ambiente operativo visuale é dotato di varie componenti. Una struttura generale fa da cornice e prende il nome di desktop. Esso si compone

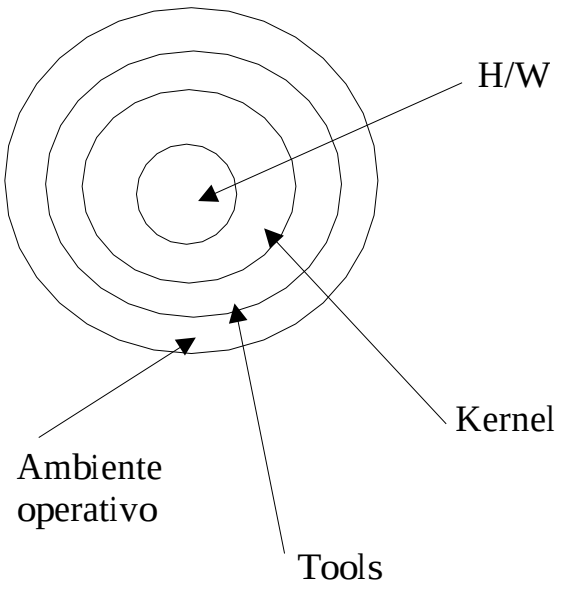

Figura 5.4: Schema di funzionamento di un sistema operativo rappresentato per livelli.

di *Area del desktop*, *Barra delle applicazioni* e *Icone*. Usualmente, dalla barra o attraverso una icona, od ancora indirettamente, dopo aver attivato un ambiente testuale, il sistema permette l'esecuzione delle componenti ulteriori:

- Il *File Manager*;
- Il *Task Manager*;
- Il *Windows Manager*.

Il File manager funge da interfaccia verso il Filesystem ed implementa in modo grafico i richiami di comandi di filesystem del sistema operativo.

Il Task Manager svolge una funzione essenzialmente identica rispetto ai comdandi di Monitoring.

Il Windows manager ha la funzione interna all'ambiente operativo visuale di amministrare le attivitá di transizione, apertura, chiusura di finestre, cioé di quelle componenti grafiche che consentono di eseguire le applicazioni e nel caso della specifica chiamata ad un ambiente testuale l'esecuzione di comandi dati in forma testuale.

5.6 Autorizzazioni e ruoli in sistemi multiutente

Ogni sistema multiutente deve necessariamente disporre di una funzione di gestione delle autorizzazioni, in particolare verso il filesystem, ed anche in specie verso le applicazioni ed i comandi stessi.

In genere si chiama *autorizzazione* o *privilegio di accesso* uno schema che dice che cosa possono fare i singoli utenti di un sistema operativo con uno specifico file o con un comando o con un'applicazione. I privilegi standard sono:

- La *lettura*;
- La *scrittura*;
- La *cancellazione*.

per il filesystem e l'*esecuzione* per comandi ed applicazioni.

Nella maggior parte dei sistemi operativi ed in generale dei sistemi multiutente[1] oltre ai generici utenti esiste almeno un utente speciale genericamente riferito come l'*Amministratore di Sistema*. Tale utente ha vari privilegi specifici, ed in specie possiede privilegio di esecuzione su alcune applicazioni inaccessibili ad ogni altro utente.

Egli puó:

- Creare e cancellare utenti;
- Leggere/scrivere e cancellare senza vincolo ogni file degli utenti;
- Eseguire senza vincolo applicazioni degli utenti.

[1] Ad esempio, Outlook della Microsoft e Lotus Notes della Lotus-IBM sono software commerciali per lagestione della posta e di basi di dati condivise che sono sistemi multiutenti ma non sistemi operativi.

Capitolo 6

Basi di programmazione e teoria degli algoritmi

6.1 Introduzione

Questo capitolo introduce due dei principali argomenti dell'Informatica: la Programmazione e la Teoria degli algoritmi. In effetti in questo capitolo ci occuperemo di argomenti preliminari e fondamentali rispetto ai temi proposti.

In particolare, il paragrafo 6.2 introduce la Forma di Backus-Naur estesa, un formato di rappresentazione della sintassi dei linguaggi di programmazione, considerata standard nella presentazione della sintassi dei linguaggi moderni.

Il paragrafo 6.3 diremo sommariamente alcuni caratteri generali dei linguaggi, in particolare specificheremo i termini*sintassi*, *semantica* (operazionale e denotazionale), *compilatore*, *interprete*, *p-code*, *run-time support*.

Il paragrafo 6.4 presenta una analoga analisi terminologica per i linguaggi orientati agli oggetti ed in specie dei tre termini chiave *classe*, *oggetto* e *metodo*.

L'ultimo paragrafo 6.5 presenta un fondamentale risultato sulla struttura dei linguaggi di programmazione, il Teorema di Jacopini-Böhm.

6.2 Forma di Backus-Naur Estesa

Definiamo la notazione detta Extended Backus-Naur Form (EBNF). Ciascuna regola nella grammatica definisce un simbolo, nella forma

symbol ::= expression

I simboli sono scritti con una lettera iniziale maiuscola se sono definiti da un'espressione regolare, o con una lettera minuscola negli altri casi. Le stringhe letterali sono tra apici. Nell'espressione della parte destra di una regola, si utilizzano le seguenti espressioni per generare (match) stringhe di uno o pi'u caratteri: ^xN dove N 'e un intero esadecimale, l'espressione corrisponde al carattere in ISO/IEC 10646 il cui valore del codice canonico (UCS-4), quando interpretato come numero binario senza segno, ha il valore indicato. Il numero degli zero iniziali nella forma ^xN 'e insignificante; Il numero degli zero iniziali nel corrispondente valore di codice 'e governato dalla codifica di carattere in uso e non 'e significativo.

In particolare

[a-zA-Z], [#xN-#xN]

corrisponde a qualsiasi carattere con un valore nel (nei) rango (ranghi) indicato(i) (inclusivo).

[^a-z], [^#xN-#xN]

corrisponde a qualsiasi carattere con un valore esterno al rango indicato.

[^abc], [^#xN#xN#xN]

corrisponde a qualsiasi carattere con un valore non compreso tra i caratteri dati.

string

corrisponde ad una stringa letterale uguale (matching) a quella data all'interno dei doppi apici.

'string'

corrisponde ad una stringa letterale uguale (matching) a quella data all'interno dei singoli apici.

Questi simboli possono essere combinati per generare (match) pattern pi'u complessi nel seguente modo, siano A e B espressioni semplici

(expression)

l'espressione 'e trattata come unit'a e pu'o essere combinata come descritto in questa lista.

A?

contiene (match) una A o niente; A opzionale.

A B

contiene una A seguita da una B.

A — B

contiene una A o una B ma non entrambi.

A - B

contiene qualsiasi stringa che corrisponda (match) ad A ma non a B.

A+

contiene una o pi'u occorrenze di A.

A*

contiene zero o pi'u occorrenze di A. Altre notazioni usate nelle produzioni sono:

/* ... */

per i commenti.

6.3 Linguaggi di programmazione

Il comando lp serve per stampare. La sintassi del comando é:

```
lp nome_file
```

Il comando lp nome_file stampa il file di nome nome_file.

6.4 Linguaggi ad oggetti

La recente storia dei linguaggi di programmazione ha visto affermarsi i seguenti modelli di linguaggio, basati sui paradigmi qui sotto definiti:

Linguaggi imperativi Si tratta di linguaggi che concepiscono la relazione tra il programma e la sua esecuzione come la relazione tra un ordine e la sua esecuzione. Il codice sorgente consiste in una sequenza di ordini.

Linguaggi funzionali I linguaggi funzionali considerano l'esecuzione di un sorgente come il calcolo di una funzione.

Linguaggi logici Questi linguaggi concepiscono l'esecuzione di un sorgente come la dimostrazione formale di una proprietá logica.

Linguaggi ad oggetti Un linguaggio ad oggetti concepisce l'esecuzione come una sequenza di passaggi di messaggio tra oggetti.

L'ultima definizione risulta meno chiara. Un linguaggio ad oggetti, come ad esempio Java, vede il proprio codice come il risultato della descrizione ad alto livello dei tipi di dato di cui lo stesso programma ha bisogno. In particolare ogni oggetto é istanza di una classe che ha un insieme di metodi associati, o in termini algebrici di operazioni associate. Quando un oggetto viene creato esso attiva i propri metodi che possono a loro volta consistere nella creazione di nuovi oggetti e nella esecuzione di loro metodi. Tutto questo processo prende generalmente il nome di message passing tra gli oggetti. Chiaramente il calcolo procede esattamente perché questo processo viene eseguito.

Le nozioni chiave che definiscono l'ambito della programmazione ad oggetti sono:

Classe Un insieme di oggetti che ammettono le stesse operazioni;

Oggetto o istanza di una classe. Un elemento di una classe;

Campo Una componente accessibile di un oggetto;

Metodo Un modo di accedere campi di un oggetto, sul piano algebrico, operazioni;

Ereditarietá La relazione esistente tra classi che si ottengono una dall'altra aggiungendo campi e metodi.

6.5 Teorema di Jacopini-Böhm

Il piú rilevante problema posto dai moderni linguaggi di programmazione imperativi ed anche da quelli ad oggetti, é la necessitá o meno di aver disponibile una istruzione di salto, ovvero una istruzione che permette di cambiare la posizione (riga) del programma che verrá eseguita successivamente.

Per comprendere la portata del Teorema di Jacopini-Böhm occorre rilevare quali siano i difetti dell'istruzione di salto:

- Impedisce di sapere con precisione quali parti del programma vengono eseguite piú di una volta;
- Obbliga il programma a conoscere le righe e la loro numerazione;
- Costringe il programmatore a strutturare l'esecuzione in sequenza multipla di una parte del programma in una complessa architettura di salti e condizionamenti al salto.

Jaopini e Böhm misero in luce come ogni programma scritto in un linguaggio di programmazione possa essere tradotto in un programma senza istruzioni di salto. In particolare essi dimostrarono che, nei flow-chart, ogni istruzione di salto puó essere eliminata ed il flow-chart stesso puøvenir riscritto usando solo tre strutture chiamate sequenza, selezione e ciclo.

Nelle figure 6.1, 6.2 e 6.3 compaiono tre diagrammi di flusso (flow-chart) per le strutture suddette.

Formalmente,

Teorema 4 (Teorema di Jacopini-Böhm)
Ogni diagramma di flusso puó essere riscritto mediante le tre sole strutture delle figure 6.1, 6.2, 6.3.

6.6 Esempi di programmazione in Java

In questo paragrafo mostreremo alcuni semplici esempi schematici della struttura del linguaggio Java. L'obiettivo non é quello di imparare a programmare con quel linguaggio, né l'apprendimento della sintassi puó essere mediato da un corpus, per quanto rilevante (e questo non lo é), di esercizi. Si tratta di prendere confidenza con:

- L'aspetto di un sorgente Java;

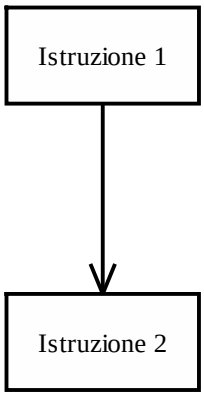

Figura 6.1: Struttura di sequenza in un diagramma di flusso.

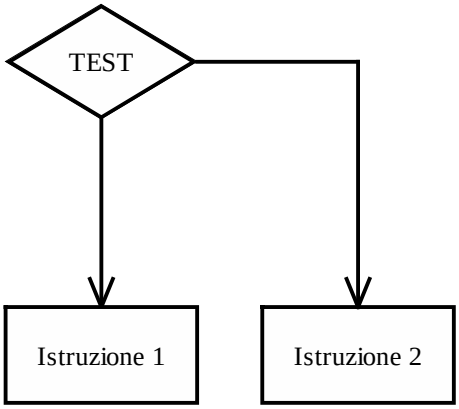

Figura 6.2: Struttura di selezione in un diagramma di flusso.

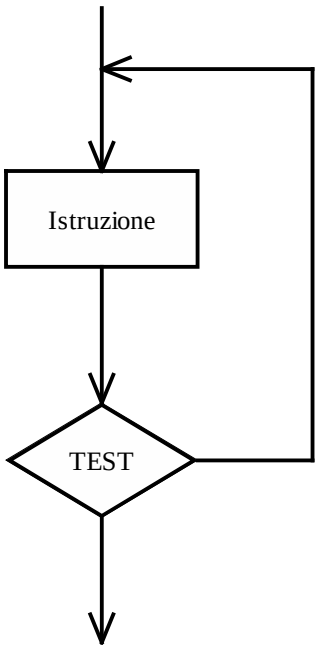

Figura 6.3: Struttura di ciclo in un diagramma di flusso.

```
class NOME_DI_CLASSE (extends NOME_DI_CLASSE);
{
   CAMPO;
   CAMPO;
   ...
   METODO;
   METODO;
   ...
}
```

Figura 6.4: Lo schema di definizione di classe in Java.

- Il comportamento di un sistema operativo quando compiliamo codice Java;
- Alcune nozioni fondamentali di Teoria degli algoritmi visitate sul codice di programmi esistenti.

Alcuni dei sorgenti qui descritti sono reperibili sul Web e gli indirizzi sono riferiti nel Capitolo 2.

In questo paragrafo descriveremo i seguenti esempi:

Definizione generale di classe La definizione di una classe molto semplice, i numeri interi, mediante la costruzione delle due operazioni elementari di somma e prodotto;

Nozione di creazione Creazione di un oggetto

Esempio di ordinamento Un esempio di implementazione in Java dell'algoritmo di ordinamento di numeri.

Definizione generale di classe La definizione generale di classe in Java si fa con l'operatore class. Lo schema é esibito in Figura 6.4.

Un esempio di classe molto semplice appare in Figura 6.5.

Nozione di creazione La creazione di uno oggetto in Java si performa mediante l'istruzione new. Essa viene usata per modificare ill significato di un assegnamento. Deve essere dato in congiunzione con la chiamata ad un operatore di creazione di oggetto, un metodo della classe a cui l'oggetto viene assegnato, come nello schema qui sotto.

```
class prova
{
/* int sta per numero intero */

    int n;

/* void sta per nessun valore */
    void prova()
        {}
    int somma (int m)
        {
        return n+m;
        }
    int product (int m)
        {
        return n*M;
        }
}
```

Figura 6.5: Un esempio di classe in Java.

```
...
prova x = new prova()
...
```

Figura 6.6: Un frammento di codice Java che esegue la creazione di un oggetto.

```
...
nome_di_classe x = new nome_di_metodo_della_classe(parametri)
...
```

Analogamente, se guardiamo alla struttura della classe prova, possiamo avere il seguente frammento di codice Java.

```
 /**
 * A bubble sort demonstration algorithm
 * SortAlgorithm.java, Thu Oct 27 10:32:35 1994
 *
 * @author James Gosling
 * @version     1.6f, 31 Jan 1995
 */
class BubbleSortAlgorithm extends SortAlgorithm {
    void sort(int a[ ]) throws Exception {
    for (int i = a.length; --i>=0; ) {
        boolean swapped = false;
        for (int j = 0; j<i; j++) {
        if (stopRequested) {
            return;
        }
        if (a[j] > a[j+1]) {
            int T = a[j];
            a[j] = a[j+1];
            a[j+1] = T;
            swapped = true;
        }
        pause(i,j);
        }
        if (!swapped)
        return;
    }
    }
}
```

Figura 6.7: Un classico algoritmo di ordinamento scritto nel linguaggio Java: Bubblesort. L'algoritmo é basato su di un generico metodo di ordinamento chiamato SortAlgorithm.

Parte II

Informatica di base - laboratorio

Capitolo 7

Comandi base di Linux per il filesystem

7.1 Accesso al sistema

Login e password Ogni utente che si connetta con un sistema UNIX o LINUX deve essere *accreditato* ovvero deve essere verificato che quella persona appartiene all'insieme degli utenti di quella macchina. Per far ció, i sistemi LINUX ed UNIX prevedono che ogni utente corrisponda ad un dato pubblico, il suo nome, ed uno privato, la sua password.

Il sistema operativo LINUX, ogni volta che non é in esecuzione alcun task di proprietá di un utente é in attesa di un accesso, cioé il sistema aspetta che un utente chieda di essere accreditato. Per ottenere l'accredito un utente di un sistema LINUX deve definire gli argomenti di due comandi parametrici *a prompt* cioé di due comandi che chiedono di valorizzare i parametri stessi quando vengono eseguiti: il comando login ed il comando password. Mentre il primo dei suddetti comandi, entro certi limiti, puó essere eseguito anche dall'utente, una volta accreditato, il secondo é un comando del solo sistema operativo. Un utente che cerchi di collegarsi vedrebbe il parametro del login liberamente mentre il parametro della password in generale é rappresentata simbolicamente da una sequenza di asterischi e risulta incomprensibile.

Una volta accreditato l'utente viene immesso in una directory speciale, la *home* dell'utente. Spesso il pathname assoluto di tale directory per un utente il cui login name sia xxxx é \home\xxxx.

Una volta accreditato, l'utente desidera uscire dal pacchetto. Il comando per fare ció é logout ovvero lo stesso login. É importante chiudere preventivamente le applicazioni aperte.

L'utente puó cambiare la propria password con il comando passwd. Il proprio identificativo di login é non modificabile.

Terminali e shell Una volta che l'accesso é avvenuto LINUX rende disponibile all'utente che si é accreditato tutti i tools del sistema operativo. Questo avviene in un ambiente operativo *verbale*, ovvero in una *shell*, oppure attraverso un ambiente *visuale*.

Dopo l'accesso infatti viene attivata una tra due *modalitá*:

- *Modalitá grafica* in cui l'accesso ai comandi avviene attraverso interazioni dirette uomo-macchina di tipo visuale;
- *Modalitá a carattere* in cui l'accesso ai comandi avviene attraverso comandi scritti.

Se la modalitá é grafica viene automaticamente attivato un ambiente operativo visuale, che viene descritto nel Capitolo **??**. Se la modalitá é a carattere viene attivato un ambiente verbale, la *shell*.

Attraverso la modalitá grafica é possibile accedere ai tools mediante una shell attivata indirettamente. Il passaggio puó essere ottenuto attraverso l'attivazione di un terminale X. In modalitá grafica esistono tipicamente dei pulsanti per l'attivazione di terminali X; ogni terminale X attiva automaticamente una shell di default, dalla quale é poi possibile transitare ad una shell diversa.

Le shell piú comuni sono:

- sh la piú elementare tra le shell esistenti, con poche caratteristiche specifiche;
- csh una shell concepita esplicitamente per l'uso con compilatori della famiglia C;
- bash una shell molto avanzata, con funzioni importanti tra cui:
  - il buffer di comandi navigabile, che permette di recuperare i comandi giá dati attraverso la freccia di movimento verso l'alto del cursore (é attiva anche la freccia verso il basso che naviga in direzione opposta - dal primo comando attivato all'ultimo);
  - la *name completion* che permette attraverso il tasto di tabulazione di cercare tra i comandi disponibili ed i file della directory corrente i file che hanno come prefisso il testo giá digitato prima di tale comando.

Il comando di attivazione di terminale é xterm.

Azioni di accesso generiche: Bootstrap All'accensione fisica della macchina viene attivato un processo di connessione tra la macchina fisica ed il sistema operativo. Parte della procedura di questo processo risiede su una porzione della memoria a sola lettura (ROM) della macchina fisica, mentre la maggior parte viene acceduta a partire da quella procedura e risiede su una unitá di I/O periferica (tipicamente un disco). Questa procedura prende il nome di *bootstrap*, ed é tramite questa procedura che un sistema operativo si impadronisce del controllo della macchina.

L'esecuzione del bootstrap non richiede nessun accesso, e porta il sistema operativo allo stato attivo senza chiedere nulla all'utente. Conseguentemente il bootstrap é una azione di accesso generica (né utente, né superutente.

Il programma chiave, che manda in esecuzione il kernel di LINUX é LILO. La configurazione del sistema operativo é definita attraverso il file di configurazione liloconf (talvolta lilo.conf).

Azioni di accesso utente: StartX Viceversa l'attivazione della modalitá grafica da parte del sistema operativo puó non essere automatica, ed in generale, quando il sistema operativo é attivo ogni utente puó attivare l modalitá grafica. I effetti, sia se il sistema operativo ha attivato automaticamente la modalitá grafica attraverso il login, sia se tale attivazione é avvenuta mediante il comando StartX, vi sono due condizioni che impediscono di iniziare la sessione grafica (sessione X):

- quando l'utente é *fuori quota* ovvero quando occupa, con i propri file, piú spazio disco di quanto stabilito per quell'account;
- quando l'utente ha giá richiesto per quell'accesso una quantitá di memoria o di tempo CPU tali da non residuare una quantitá sufficiente per l'esecuzione del server grafico X.

Azioni di accesso superutente: Shutdown Il superutente puó stabilire di concludere la sessione attiva attraverso un comando di conclusione sessione shutdown. Perché l'effetto dello shutdown sia immediato occorre eseguire, in particolare shutdown -halt now.

7.2 Organizzazione e movimento nel filesystem: pathname assoluti e relativi

File e directory Il filesystem di un sistema operativo é l'insieme di tutti i dati archiviati sui supporti di memoria persistente, e nel contesto di Linux, anche degli oggetti che costituiscono la struttura hardware del sistema stesso, inclusi i dispositivi.

In particolare Linux, cosícome Unix, distingue tre categorie di oggetti che possono appartenere al filesystem:

- i *file*;
- le *directory*;
- i *device*.

Un file é un archivio di dati, che, dal punto di vista tipologico puó essere descritto sulla base dei seguenti criteri:

- formato;
- relazione con il kernel del s.o.;
- relazione con i tools del s.o. o con le applicazioni e l'ambiente operativo.

Dal punto di vista del formato i file si distinguono in *ascii* e *binari*. I primi sono scritti in codice ASCII, ANSI o UNICODE, ed in ogni caso sono file di tipo testuale che possono essere interpretati byte per byte. I secondi invece richiedono una interpretazione per parola (tipicamente, per le macchine attuali a 32 o 64 bit - 4 o 8 byte).

Dal punto di vista della relazione con il kernel i file si distinguono tra eseguibili e non eseguibili. Alcuni file di formato ascii possono essere eseguiti, in specie le sequenze di comandi dell'ambiente operativo testuale detti *script di shell* possono essere eseguiti.

Dal punto di vista della relazione con gli eseguibili (tools, applicazioni e ambiente operativo), se questa relazione esiste, i file possono essere di due tipi:

- file (in genere binari, ma non necessariamente) scritti con una applicazione (come i fogli di Excel), o in funzione di una applicazione (come i sorgenti Java). In questo caso i file vengono detti *documenti*;
- *file di appoggio* per una applicazione come i file di inizializzazione, di configurazione o di supporto all'esecuzione.

Una directory viceversa é un contenitore di oggetti dei tre tipi su elencati (file, directory, device). L'intero filesystem di un s.o. Linux o Unix é una singola directory detta *root* ed indicata con /.

Un device é invece identificativo di una periferica, sia disco che d'altro tipo (stampanti, scanner, mouse, ed ogni altro tipo di dispositivo periferico).

Directory corrente Attraverso il comando CD che permette di muoversi attraverso il filesystem si puó cambiare la posizione assunta da ciascun singolo accesso al kernel (un terminale) nel filesystem stesso. Per ogni accesso questa posizione corrisponde ad una directory, nota con il nome di directory corrente.

La directory corrente definisce anche una specifica *variabile d'ambiente* ovvero una variabile di ciascun ambiente testuale attivato. Se accediamo al kernel mediante un terminale questo accesso valorizza una variabile dell'ambiente operativo che corrisponde alla posizione nel filesystem in cui "ci si trova".

La definizione della variabile permette di dar significato ai nomi speciali di variabile . e .. che rispettivamente identificano la directory corrente e la directory padre della directory corrente.

Pathname assoluti Un *pathname* é un nome di sistema operativo che identifica un file nel filesystem. Esso é composto di due parti:

- il nome della directory contenitore;
- il nome del file o della directory definita dal nome stesso.

In particolare chiamiamo pathname assoluto un nome in cui il primo simbolo del nome della directory contenitore é il simbolo di root /.

Un pathname assoluto identifica il file o la directory il cui nome é separato a sinistra dal simbolo / dal nome della directory contenitore. Per conformitá, dunque, un pathname assoluto é una sequenza finita di nomi validi di directory separati da /, tali per cui ogni coppia in sequenza esiste nel filesystem ed é in relazione da padre (a sinistra) a figlio (a destra).

Ad esempio se un albero di filesystem fosse quello disegnato in Figura 7.1, i pathname assoluti delle directory A, B, C, D ed E sarebbero rispettivamente:

A. Pathname /A;

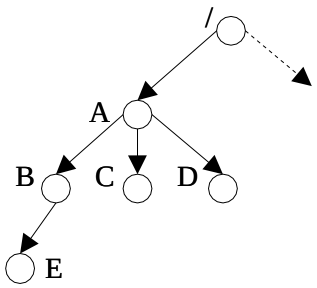

Figura 7.1: Un esempio di filesystem.

B. Pathname $/A/B$;

C. Pathname $/A/C$;

D. Pathname $/A/D$;

E. Pathname $/A/B/E$.

Pathname relativi Un pathname relativo identifica viceversa il file o la directory il cui nome **non** é separato a sinistra dal simbolo / dal nome della directory contenitore. Inoltre tra i nomi di directory possono apparire i simboli di directory corrente (.) e di directory padre (..). Per conformitá, dunque, un pathname assoluto é una sequenza finita di nomi validi di directory (anche simbolici) separati da /, tali per cui ogni coppia in sequenza esiste nel filesystem ed é in relazione da padre (a sinistra) a figlio (a destra) oppure uno dei due od entrambi gli elementi di tale coppia sono simboli di directory corrente o padre.

Un pathname relativo viene interpretato come la compattazione (ottenuta eliminando i percorsi intermedi che sono circolari) del pathname assoluto generato aggiungendo a sinistra il pathname assoluto della directory corrente al pathname assoluto determinato dalla sostituzione dei simboli di directory corrente e padre della directory corrente nel pathname relativo. Ad esempio la directory E potrá essere individuata a partire da una directory con i seguenti pathname relativi:

- A partire da C: $../B/E$;
- A partire da D: $../B/E$;

- A partire da B: E;
- A partire da A: B/E.

Comando di movimento cd Il comando cd ha un unico argomento, la directory obiettivo. L'esecuzione del comando ha come effetto la valorizzazione della variabile d'ambiente che identifica la directory corrente con l'argomento del comando.

Il comando richiede necessariamente l'argomento, cosicché se il comando viene chiamato senza argomento l'esecuzione non avviene.

Un caso comune d'uso é la combinazione cd .. il cui effetto é il riposizionamento della directory corrente sulla directory padre della precedente directory corrente.

Il comando cd é ovviamente inefficace quando la directory obiettivo é specificata in modo inconsistente, o quando non esiste. É rilevante notare che poiché la directory corrente é una variabile d'ambiente della shell, se il comando cd é in esecuzione in una shell per esempio su di un terminale X aperto (chiamato terminale A) e simultaneamente un secondo terminale X é aperto (terminale B), se un comando di rimozione directory viene eseguito sulla directory corrente di A da B, in A la variabile diretory corrente risulterá indefinita, ed occorrerrá un comando cd che rivalorizzi la variabile correttamente.

Se il comando cd viene chiamato senza argomento la directory home dell'account che esegue il comando diviene la directory corrente.

7.3 Creazione e rimozione di Directory

Il comando mkdir Il comando mkdir in LINUX crea una directory vuota. Ha un solo argomento, il pathname della directory da creare.

I vincoli alla creazione di una directory sono molto semplici. Occorre infatti solamente che la directory che viene creata non esista giá, ovvero che il pathname assoluto ottenuto traducendo il pathname argomento non corrisponda a nessuna directory esistente.

Il comando rmdir Il comando rmdir di LINUX ha un solo argomento, la directory da rimuovere.

Il comando rmdir di LINUX rimuove una directory D purché:

- D sia vuota;

- la directory corrente non sia D.

7.4 Visualizzazione del contenuto di una directory

Il comando ls Il comando ls ha un unico argomento opzionale che deve essere il pathname di una directory. Il comando elenca il contenuto del suo argomento. Se ls viene chiamato senza argomento elenca il contenuto della directory corrente.

Due opzioni di ls sono importanti nella pratica:

- -a che permette di elencare anche i file nascosti, cioé quei file il cui nome locale inizia con il simbolo .;
- -l che elenca il contenuto con i dettagli, in particolare indicando privilegi di accesso, dimensione, data di creazione e proprietario.

Il comando pwd Il comando pwd privo di argomenti, ritorna il pathname assoluto della directory corrente.

7.5 Utenti, Privilegi di accesso e loro definizione

Il comando who Il comando who prende un argomento opzionale che specifica l'indirizzo di una macchina del dominio Linux del server a cui la macchina da cui si é connessi fa riferimento. Se l'argomento é assente, il comando elenca gli utenti connessi con il sistema. Il comando elenca ogni connessione indicando il nome dell'utente connesso e l'identificativo del terminale da cui avviene la connessione. Se la connessione avviene dalla stessa macchina in genere questo viene indicato con il nome *console*. Se la connessione avviene in modo remoto viene indicato il comando di connessione telnet, rlogin, ssh.

Se l'argomento é presente il comando ritorna gli stessi dati del caso in cui il comando venga digitato senza argomenti riferiti alla macchina del dominio specificata.

Il comando whoami Il comando whoami non prende argomenti e ritorna il nome dell'utente connesso che digita il comando stesso. Questo comando puó apparire inutile, ma poiché é perfettamente possibile effettuare piú connessioni da parte dello stesso individuo con piú nomi di utente, e potrebbe averlo dimenticato.

Il comando chmod Il comando chmod permette di modificare i privilegi di accesso di un file o di una directory. Lo schema é molto semplice, ed é in base ottale.

- Il numero 1 corrisponde al privilegio di lettura;
- Il numero 2 corrisponde al privilegio di scrittura;
- Il numero 4 corrisponde al privilegio di esecuzione.

Se ad un file desideriamo dare, per un gruppo di utenti, il privilegio di lettura e scrittura, diamo il numero di codifica 3 (1+2), per la lettura ed esecuzione diamo 5 (1+4), per la scrittura ed esecuzione 6 (2+4), per tutti i privilegi 7 (1+2+4), per nessun privilegio 0.

I sistemi Linux individuano tre categorie di utenti cui attribuire i privilegi su elencati:

- **owner**, il proprietario del file;
- **group**, il gruppo di utenti cui il proprietario appartiene;
- **world**, l'insieme di tutti gli utenti del sistema.

La codifica del comando chmod viene data come primo argomento dei due obbligatori, mentre il secondo é il file a cui vengono attribuiti i privilegi stessi.

Ad esempio, per permettere a tutti gli utenti di compiere qualsiasi operazione su di un particolare file xxxx, occorre il comando

```
chmod 777 xxxx
```

Viceversa per riservarsi ogni diritto negandolo a tutti coloro i quali non sono propritari di un certo file occorre il comando

```
chmod 700 xxxx
```

7.6 Rimozione, spostamento, copia di file

Il comando rm Il comando rm rimuove un file il cui nome sia specificato. Prende un unico parametro, il pathname del file da cancellare. Non funziona se il file da cancellare viene specificato in modo errato e se colui il quale esegue il comando non ha il privilegio per cancellare quel file.

Il comando mv Il comando mv rimuove un file il cui nome sia specificato e lo copia in un secondo file. Prende due parametri, il pathname del file da spostare ed il pathname di un file o di una directory . Non funziona se il file da spostare o il luogo dove spostarlo vengono specificati in modo errato, se colui il quale esegue il comando non ha il privilegio per cancellare quel file oppure se il file viene spostato su di un nome di file che giá esiste.

Il comando cp Il comando cp copia un file in un altro. Prende due argomenti, sorgente e destinazione. Non funziona se la sorgente e la destinazione sono uguali o se la directory dei pathname non esiste.

Se la destinazione é una directory il file viene copiato in quella directory con lo stesso nome della sorgente.

Capitolo 8

Comandi base di Linux per il monitor

8.1 Il comando ps

Il comando ps lista i processi in esecuzione da parte dell'utente sulla macchina. Normalmente non prende argomenti.

Due parametri sono importanti per ps:

- -l, che lista i dettagli dei processi, compresi il proprietario, il PID (Process Identification Number) ed il tempo di inizio;
- -a, che lista tutti i processi in esecuzione sulla macchina.

8.2 Il comando kill

Il comando kill invia un segnale ad un processo in esecuzione. Prende come parametro il **PID** (Process Identification Number) numero di identificazione del processo. Una opzione importante é il numero identificativo del tipo di segnale da inviare.

Per gli scopi di questo corso l'unico segnale di interesse é il numero 9, che consiste in un interrupt seguito da uno stop. L'effetto é che il processo a cui il segnale é stato inviato termina.

In genere la sequenza necessaria perché un processo che disturba (per la pesantezza d'esecuzione) venga interrotto é di listare i processi con l'opzione -l di ps e poi inviare il segnale -9 a quel processo, identificandolo con il PID.

In pratica,

```
ps -l
```

Ora il sistema elenca i processi ed il PID del nostro candidato alla chiusura é 1955.

```
kill -9 1955
```

Capitolo 9

Comandi base di Linux per il Networking

9.1 Comandi di identificazione del network

Il comando ping Al comando ping é necessario far seguire il nome di un computer remoto. Il comando ping serve per verificare, tramite la rete, lo stato di un computer. Una volta lanciato, il programma ping spedisce dei dati al computer indicato e aspetta la risposta. Valutando tale risposta, si puó stabilire se il computer funziona correttamente, se é spento oppure se é sovraccarico.

```
ping www.univr.it
```

Il comando finger Il comando finger é utile nel caso in cui si voglia verificare se un certo utente é connesso ad una determinata macchina collegata in rete. La sintassi del comando é :

```
finger nome_utente@nome_computer
```

9.2 Comandi di trasferimento del controllo di esecuzione

Il comando telnet Il comando telnet serve per collegarsi ad un computer remoto. Tramite il programma telnet é possibile eseguire comandi e programmi su un computer che non é quello su cui si sta lavorando. La sintassi del comando é :

```
telnet nome_computer
```

Una volta lanciato il comando, viene richiesto di autenticarsi con nome utente e password. Serve, quindi, un account valido sul computer remoto.

Il comando ssh Il comando ssh serve per collegarsi ad un computer remoto. Tramite il programma ssh é possibile eseguire comandi e programmi su un computer che non é quello su cui si sta lavorando. Il suo funzionamento é simile al programma telnet. La differenza consiste nel fatto che i dati trasmessi sulla rete con il comando ssh sono criptati e, quindi, difficilmente intercettabili. La sintassi del comando é :

```
ssh nome_computer
```

Una volta lanciato il comando, viene richiesto di autenticarsi con nome utente e password. Serve, quindi, un account valido sul computer remoto.

Capitolo 10

Comandi base di Linux per la gestione delle periferiche

10.1 Comandi per la messa a disposizione di periferiche

Il comando mount Il comando mount ha come parametri il nome di una periferica (opzionale) e il nome di una directory. La sintassi del comando é:

```
mount nome_periferica nome_directory
```

Il comando mount serve per accedere al filesystem presente nella periferica a cui si fa riferimento. Digitando mount nome_periferica nome_directory il kernel di Linux rende disponibile all'interno della directory nome_directory il contenuto del filesystem presente nella periferica nome_periferica.

Il nome della periferica opzionale. Nella directory /etc é presente un file di nome fstab. In questo file sono specificati i nomi delle periferiche che generalmente sono utilizzate e i nomi delle directory in cui sono presenti i relativi filesystem. Se nel file fstab alla periferica nome_periferica é associata la directory nome_directory si puó omettere il nome della periferica e scrivere solo mount nome_directory. Ad esempio, nel file fstab alla periferica /dev/fd0 (il floppy drive) normalmente é associata la directory /mnt/floppy. Quindi per accedere ai file presenti nel floppy disk si puó scrivere:

```
mount /dev/fd0 /mnt/floppy
```

Analogamente risulta valida la sintassi:

```
mount /mnt/floppy
```

Il comando mount normalmente é in grado di riconoscere automaticamente il tipo di filesystem presente nella periferica. Tuttavia é possibile specificarlo manualmente mediante l'opzione -t. Scrivendo mount -t tipo_filesystem nome_periferica nome_directory si forza il sistema operativo a credere che nella periferica nome_periferica ci sia il filesystem di tipo tipo_filesystem. I tipi di filesystem piú diffusi sono:

msdos, il filesystem usato dal vecchio ms-dos
vfat, il filesystem usato da windows 98, me
ntfs, il filesystem usato da windows NT, 2000 e XP home e professional
ext2, il filesystem usato da Linux
ext3, un filesystem evoluto usato da Linux
iso9660, il filesystem usato nei cdrom
...

Il comando umount Il comando umount ha come parametri il nome di una directory. La sintassi del comando é:

```
umount nome_directory
```

Il comando umount serve per rimuovere un filesystem montato precedentemente. Digitando umount nome_directory il kernel di Linux elimina l'accesso al filesystem presente nella periferica che era precedentemente stata montata sulla directory nome_directory. Questo non significa che il file presenti in quel filesystem sono stati eliminati, ma che non sono piú raggiungibili, a meno di non rimontare il filesystem con il comando mount.

Ad esempio, dopo aver utilizzato i file presenti nel floppy disk, prima di estrarre il dischetto, si deve effettuare l'umount della directory /mnt/floppy (supponendo che il floppy disk sia stato montato lí). Quindi si deve digitare:

```
umount /mnt/floppy
```

É molto importante effettuare l'umount di una periferica prima di toglierla dal computer per evitare di perdere i propri dati. Linux, infatti, per migliorare l'accesso alle periferiche, salva in memoria le modifiche apportate ai file e le scrive sulla periferica solo quando non sta lavorando. Quindi, se si estrae un floppy disk prima di aver effettuato l'umount, é possibile che si perdano i dati che erano stati scritti nel frattempo sul floppy disk.

10.2 Comandi di stampa

Il comando lp Il comando lp serve per stampare. La sintassi del comando é:

```
lp nome_file
```

Il comando lp nome_file stampa il file di nome nome_file.

Il comando lprm Il comando lprm serve per rimuovere ogni documento dalla coda di stampa. La sintassi del comando é:

```
lprm
```

Se per qualche motivo si é lanciata una stampa di un documento con il comando lp nome_file e si vuole eliminare la stampa, basta digitare lprm. Con questo comando non si cancella il file, ma si dice al sistema che non deve piú stamparlo.

10.3 Il pacchetto mstools

Il comando mdir Il comando mdir serve per ottenere un elenco dei file e delle directory presenti nel floppy disk. La sintassi del comando é :

```
mdir a:/nome_directory
```

Se si esegue il comando mdir senza parametri il programma restituisce l'elenco di file e directory presenti nella directory principale del floppy disk, altrimenti, se viene lanciato con il parametro a:/nome_directory, il programma mostra solo il contenuto della directory nome_directory.

Il comando mcopy Il comando mcopy serve per copiare uno o piú file da o nel floppy disk. La sintassi del comando é:

```
mcopy a:/directory_origine/file_origine
      directory_destinazione/file_destinazione
```

Digitando mcopy a:/file_origine directory_destinazione/file_destinazione, il programma mcopy copia dal floppy disk il file_origine all'interno della directory chiamata directory_destinazione nominandolo file_destinazione. É possibile omettere il nome del file destinazione, in tal caso tale file prende il nome del file di origine.

Una sintassi alternativa ammessa é:

```
mcopy directory_origine/nome_file_origine
      a:/directory_destinazione/nome_file_destinazione
```

In questo caso si copia il file nome_file_origine all'interno della directory directory_destinazione nel floppy drive

Ad esempio, per copiare il file pippo dal floppy nella directory corrente basta digitare il comando:

```
mcopy a:/pippo .
```

Nota: '.' identifica la directory corrente (qualsiasi essa sia).

Il comando mdel Il comando mdel serve per eliminare uno o piú file dal floppy disk. La sintassi del comando é :

```
mdel a:/nome_file1 nome_file2 ...
```

Ad esempio, per eliminare il file prova dal floppy basta digitare il comando:

```
mdel a:/prova
```

Il comando mdeltree Il comando mdeltree é simile al comando mdel. A differenza di mdel, mdeltree elimina ricorsivamente una directory ed il suo contenuto dal floppy disk. La sintassi del comando é:

```
mdeltree a:/nome_directory1 nome_directory2 ...
```

Ad esempio, per eliminare la directory dir_prova e tutto il suo contenuto dal floppy basta digitare il comando:

```
mdeltree a:/dir_prova
```

Il comando mformat Il comando mformat crea un filesystem di tipo msdos nel floppy disk eliminado qualsiasi dato fosse contenuto nel floppy disk. É l'equivalente del comando format a: lanciato da un terminale dos. La sintassi del comando é :

```
mformat
```

É importante sapere che quando si lancia il comando mformat si perdono tutti i dati precedentemente contenuti nel floppy disk.

Capitolo 11

Comandi di Filesystem del sistema Windows

11.1 Esecuzione di comandi di visualizzazione dal prompt di MS-DOS

Il comando di MS-DOS dir richiede un argomento, la directory il cui contenuto deve essere elencato. Il comando restituisce la lista dei file e delle sottodirectory incluse nella directory argomento.

Le due pi rilevanti opzioni sono /p che esegue il comando in modalit con pausa, vale a dire interrompendo e rieseguendo il comando su pressione di un tasto ogni volta che lo scroll dello schermo ha reso invisibili le ultime righe, e /w che visualizza multicolonna la lista evitando i dettagli.

11.2 Creazione e cancellazione directory

Il comando mkdir in WINDOWS crea una directory vuota. Ha un solo argomento, il pathname della directory da creare.

I vincoli alla creazione di una directory sono analoghi a quelli posti al comando corrispondente in Linux. Occorre infatti solamente che la directory che viene creata non esista giá, ovvero che il pathname assoluto ottenuto traducendo il pathname argomento non corrisponda a nessuna directory esistente.

La cancellazione di una directory si performa con il comando rmdir il cui comportamento é analogo a quello del comando corrispondente di Lin-

ux. Il comando mkdir puó abbreviarsi in md, mentre il comando rmdir puó abbreviarsi in rd.

11.3 Cancellazione, copia e spostamento di file

Il comando del serve per eliminare uno o piú file in Windows. La sintassi del comando é :

```
del nome_file1 nome_file2 ...
```

Ad esempio, per eliminare il file pippo basta digitare il comando:

```
del pippo
```

Si noti che l'eliminazione di piú files puó essere performata mediante l'uso delle wild cards. Se, in particolare, si usa il comando

```
del pippo.*
```

il cui effetto é l'eliminazione di tutti i file che hanno estensione qualsiasi e nome pippo. Viceversa, il comando

```
del pipp?
```

elimina tutti i file il cui nome inizia con pipp seguito da una singola lettera.

Appendice A

Classi di caratteri in standard Unicode e Unicode-RTF

Seguendo le caratteristiche definite nello standard Unicode, i caratteri cono classificati come caratteri di base (tra gli altri, questi contengono i caratteri alfabetici dell'alfabeto Latino, senza i diacritici), caratteri ideografici, e caratteri che si combinano (tra gli altri, questa classe contiene la maggior parte dei diacritici); queste classi si combinano a formare la classe delle lettere. Cifre e estensori sono anche distinti.

```
Letter::= BaseChar | Ideographic

BaseChar::=

[#x0041-#x005A]  |[#x0061-#x007A] |[#x00C0-#x00D6] |[#x00D8-#x00F6]
|[#x00F8-#x00FF] |[#x0100-#x0131] |[#x0134-#x013E] |[#x0141-#x0148]
|[#x014A-#x017E] |[#x0180-#x01C3] |[#x01CD-#x01F0] |[#x01F4-#x01F5]
|[#x01FA-#x0217] |[#x0250-#x02A8] |[#x02BB-#x02C1] |#x0386
|[#x0388-#x038A] |#x038C          |[#x038E-#x03A1] |[#x03A3-#x03CE]
|[#x03D0-#x03D6] |#x03DA |#x03DC  |#x03DE |#x03E0  |[#x03E2-#x03F3]
|[#x0401-#x040C] |[#x040E-#x044F] |[#x0451-#x045C] |[#x045E-#x0481]
|[#x0490-#x04C4] |[#x04C7-#x04C8] |[#x04CB-#x04CC] |[#x04D0-#x04EB]
|[#x04EE-#x04F5] |[#x04F8-#x04F9] |[#x0531-#x0556] |#x0559
|[#x0561-#x0586] |[#x05D0-#x05EA] |[#x05F0-#x05F2] |[#x0621-#x063A]
```

```
|[#x0641-#x064A] |[#x0671-#x06B7] |[#x06BA-#x06BE] |[#x06C0-#x06CE]
|[#x06D0-#x06D3] |#x06D5          |[#x06E5-#x06E6] |[#x0905-#x0939]
|#x093D          |[#x0958-#x0961] |[#x0985-#x098C] |[#x098F-#x0990]
|[#x0993-#x09A8] |[#x09AA-#x09B0] |#x09B2          |[#x09B6-#x09B9]
|[#x09DC-#x09DD] |[#x09DF-#x09E1] |[#x09F0-#x09F1] |[#x0A05-#x0A0A]
|[#x0A0F-#x0A10] |[#x0A13-#x0A28] |[#x0A2A-#x0A30] |[#x0A32-#x0A33]
|[#x0A35-#x0A36] |[#x0A38-#x0A39] |[#x0A59-#x0A5C] |#x0A5E
|[#x0A72-#x0A74] |[#x0A85-#x0A8B] |#x0A8D          |[#x0A8F-#x0A91]
|[#x0A93-#x0AA8] |[#x0AAA-#x0AB0] |[#x0AB2-#x0AB3] |[#x0AB5-#x0AB9]
|#x0ABD |#x0AE0  |[#x0B05-#x0B0C] |[#x0B0F-#x0B10] |[#x0B13-#x0B28]
|[#x0B2A-#x0B30] |[#x0B32-#x0B33] |[#x0B36-#x0B39] |#x0B3D
|[#x0B5C-#x0B5D] |[#x0B5F-#x0B61] |[#x0B85-#x0B8A] |[#x0B8E-#x0B90]
|[#x0B92-#x0B95] |[#x0B99-#x0B9A] |#x0B9C          |[#x0B9E-#x0B9F]
|[#x0BA3-#x0BA4] |[#x0BA8-#x0BAA] |[#x0BAE-#x0BB5] |[#x0BB7-#x0BB9]
|[#x0C05-#x0C0C] |[#x0C0E-#x0C10] |[#x0C12-#x0C28] |[#x0C2A-#x0C33]
|[#x0C35-#x0C39] |[#x0C60-#x0C61] |[#x0C85-#x0C8C] |[#x0C8E-#x0C90]
|[#x0C92-#x0CA8] |[#x0CAA-#x0CB3] |[#x0CB5-#x0CB9] |#x0CDE
|[#x0CE0-#x0CE1] |[#x0D05-#x0D0C] |[#x0D0E-#x0D10] |[#x0D12-#x0D28]
|[#x0D2A-#x0D39] |[#x0D60-#x0D61] |[#x0E01-#x0E2E] |#x0E30
|[#x0E32-#x0E33] |[#x0E40-#x0E45] |[#x0E81-#x0E82] |#x0E84
|[#x0E87-#x0E88] |#x0E8A |#x0E8D  |[#x0E94-#x0E97] |[#x0E99-#x0E9F]
|[#x0EA1-#x0EA3] |#x0EA5 |#x0EA7  |[#x0EAA-#x0EAB] |[#x0EAD-#x0EAE]
|#x0EB0 |[#x0EB2-#x0EB3] |#x0EBD  |[#x0EC0-#x0EC4] |[#x0F40-#x0F47]
|[#x0F49-#x0F69] |[#x10A0-#x10C5] |[#x10D0-#x10F6] |#x1100
|[#x1102-#x1103] |[#x1105-#x1107] |#x1109          |[#x110B-#x110C]
|[#x110E-#x1112] |#x113C |#x113E  |#x1140 |#x114C  |#x114E |#x1150
|[#x1154-#x1155] |#x1159          |[#x115F-#x1161] |#x1163 |#x1165
|#x1167 |#x1169  |[#x116D-#x116E] |[#x1172-#x1173] |#x1175 |#x119E
|#x11A8 |#x11AB  |[#x11AE-#x11AF] |[#x11B7-#x11B8] |#x11BA
|[#x11BC-#x11C2] |#x11EB |#x11F0  |#x11F9          |[#x1E00-#x1E9B]
|[#x1EA0-#x1EF9] |[#x1F00-#x1F15] |[#x1F18-#x1F1D] |[#x1F20-#x1F45]
|[#x1F48-#x1F4D] |[#x1F50-#x1F57] |#x1F59 |#x1F5B  |#x1F5D
|[#x1F5F-#x1F7D] |[#x1F80-#x1FB4] |[#x1FB6-#x1FBC] |#x1FBE
|[#x1FC2-#x1FC4] |[#x1FC6-#x1FCC] |[#x1FD0-#x1FD3] |[#x1FD6-#x1FDB]
|[#x1FE0-#x1FEC] |[#x1FF2-#x1FF4] |[#x1FF6-#x1FFC] |#x2126
|[#x212A-#x212B] |#x212E          |[#x2180-#x2182] |[#x3041-#x3094]
|[#x30A1-#x30FA] |[#x3105-#x312C] |[#xAC00-#xD7A3]

Ideographic::=
```

```
[#x4E00-#x9FA5] |#x3007 |[#x3021-#x3029]

CombiningChar::=

[#x0300-#x0345]  |[#x0360-#x0361]  |[#x0483-#x0486] |[#x0591-#x05A1]
|[#x05A3-#x05B9] |[#x05BB-#x05BD]  |#x05BF          |[#x05C1-#x05C2]
|#x05C4          |[#x064B-#x0652]  |#x0670          |[#x06D6-#x06DC]
|[#x06DD-#x06DF] |[#x06E0-#x06E4]  |[#x06E7-#x06E8] |[#x06EA-#x06ED]
|[#x0901-#x0903] |#x093C           |[#x093E-#x094C] |#x094D
|[#x0951-#x0954] |[#x0962-#x0963]  |[#x0981-#x0983] |#x09BC |#x09BE
|#x09BF          |[#x09C0-#x09C4]  |[#x09C7-#x09C8] |[#x09CB-#x09CD]
|#x09D7          |[#x09E2-#x09E3]  |#x0A02 |#x0A3C  |#x0A3E |#x0A3F
|[#x0A40-#x0A42] |[#x0A47-#x0A48]  |[#x0A4B-#x0A4D] |[#x0A70-#x0A71]
|[#x0A81-#x0A83] |#x0ABC           |[#x0ABE-#x0AC5] |[#x0AC7-#x0AC9]
|[#x0ACB-#x0ACD] |[#x0B01-#x0B03]  |#x0B3C          |[#x0B3E-#x0B43]
|[#x0B47-#x0B48] |[#x0B4B-#x0B4D]  |[#x0B56-#x0B57] |[#x0B82-#x0B83]
|[#x0BBE-#x0BC2] |[#x0BC6-#x0BC8]  |[#x0BCA-#x0BCD] |#x0BD7
|[#x0C01-#x0C03] |[#x0C3E-#x0C44]  |[#x0C46-#x0C48] |[#x0C4A-#x0C4D]
|[#x0C55-#x0C56] |[#x0C82-#x0C83]  |[#x0CBE-#x0CC4] |[#x0CC6-#x0CC8]
|[#x0CCA-#x0CCD] |[#x0CD5-#x0CD6]  |[#x0D02-#x0D03] |[#x0D3E-#x0D43]
|[#x0D46-#x0D48] |[#x0D4A-#x0D4D]  |#x0D57 |#x0E31  |[#x0E34-#x0E3A]
|[#x0E47-#x0E4E] |#x0EB1           |[#x0EB4-#x0EB9] |[#x0EBB-#x0EBC]
|[#x0EC8-#x0ECD] |[#x0F18-#x0F19]  |#x0F35 |#x0F37  |#x0F39 |#x0F3E
|#x0F3F          |[#x0F71-#x0F84]  |[#x0F86-#x0F8B] |[#x0F90-#x0F95]
|#x0F97          |[#x0F99-#x0FAD]  |[#x0FB1-#x0FB7] |#x0FB9
|[#x20D0-#x20DC] |#x20E1           |[#x302A-#x302F] |#x3099 |#x309A

Digit::=

[#x0030-#x0039]  |[#x0660-#x0669] |[#x06F0-#x06F9] |[#x0966-#x096F]
|[#x09E6-#x09EF] |[#x0A66-#x0A6F] |[#x0AE6-#x0AEF] |[#x0B66-#x0B6F]
|[#x0BE7-#x0BEF] |[#x0C66-#x0C6F] |[#x0CE6-#x0CEF] |[#x0D66-#x0D6F]
|[#x0E50-#x0E59] |[#x0ED0-#x0ED9] |[#x0F20-#x0F29]


Extender::=

#x00B7           |#x02D0 |#x02D1  |#x0387 |#x0640 |#x0E46 |#x0EC6 |#x3005
|[#x3031-#x3035] |[#x309D-#x309E] |[#x30FC-#x30FE]
```

Le classi di carattere definite qui possono essere derivate dall'archivio dei caratteri Unicode come segue: I caratteri che iniziano i nomi devono avere una delle categorie Ll, Lu, Lo, Lt, Nl. I caratteri dei nomi oltre che i caratteri che iniziano i nomi devono avere una delle categorie Mc, Me, Mn, Lm, or Nd. I caratteri nell'area di compatibilitá (cioé con codice di carattere maggiore di #xF900 e minore di #xFFFE) non sono consentiti nei nomi. I caratteri che hanno un font o una decomposizione di compatibilitá (cioé quelli con un tag di formattazione di compatibilitá nel campo 5 dell'archivio – segnato dal campo 5 che inizia con un «) non sono permessi.

I seguenti caratteri sono trattati come caratteri che iniziano i nomi piuttosto che come caratteri nome, poiché il file di proprietá li classifica come Alfabetici: [#x02BB-#x02C1], #x0559, #x06E5, #x06E6. I caratteri #x20DD-#x20E0 sono esclusi (in accordo con Unicode, sezione 5.14).

Il carattere x00B7 é classificato come estensore, poiché la lista di proprietá lo identifica cos. Il carattere #x0387 é aggiunto come nome carattere, poiché #x00B7 é il suo equivalente canonico. I caratteri ':' and '_' sono consentiti come caratteri che iniziano i nomi. I caratteri '−' and '.' sono consentiti come caratteri nomi.

Appendice B

Codici ASCII e codici ANSI

ASCII é l'acronimo per American Standard Code for Information Interchange (pron. aski), codice americano standard per l'interscambio delle informazioni. Approvato nel 1968 dall'ANSI, il codice ASCII standard e' supportato praticamente da tutti i produttori di computer per rappresentare lettere maiuscole, minuscole, numeri, punteggiatura e caratteri speciali. Ciascuno dei 128 caratteri ha il proprio codice detto codice ASCII ed ogni carattere occupa 7 bit all'interno di un byte. Rimanendo inutilizzato l'ottavo bit, alcuni produttori lo hanno impiegato per definire 128 nuovi caratteri (ASCII esteso con caratteri grafici, lettere nazionali, ecc.) creando versioni diverse e talvolta incompatibili La tabella di codice ASCII esteso internazionale approvata da ANSI nel 1998 appare in tabella B. Le caselle vuote sono riempibili con caratteri speciali che dipendono dal linguaggio. La tabella qui presentata é la parte comune a tutte le versioni.

N.	S.	N.	S.	N.	S.	N.	S.	N.	S.	N.	S.
0	NUL	1	c−A	2	c−B	3	c−C	4	c−D	5	c−E
6	c−F	7	c−G	8	c−H	9	c−I	10	c−J	11	c−K
12	c−L	13	c−M	14	c−N	15	c−O	16	c−P	17	c−Q
18	c−R	19	c−S	20	c−T	21	c−U	22	c−V	23	c−W
24	c−X	25	c−Y	26	c−Z	27	c−[	28	c−\	29	c−]
30	c−^	31	c−	32		33	!	34	”	35	#
36	$	37	%	38	&	39	′	40	(	41	)
42	∗	43	+	44	,	45	−	46	.	47	/
48	0	49	1	50	2	51	3	52	4	53	5
54	6	55	7	56	8	57	9	58	:	59	;
60	<	61	=	62	>	63	?	64	@	65	A
66	B	67	C	68	D	69	E	70	F	71	G
72	H	73	I	74	J	75	K	76	L	77	M
78	N	79	O	80	P	81	Q	82	R	83	S
84	T	85	U	86	V	87	W	88	X	89	Y
90	Z	91	[	92	\	93	]	94	^	95	_
96	‘	97	a	98	b	99	c	100	d	101	e
102	f	103	g	104	h	105	i	106	j	107	k
108	l	109	m	110	n	111	o	112	p	113	q
114	r	115	s	116	t	117	u	118	v	119	w
120	x	121	y	122	z	123	–	124	|	125	″
126	~	127	□	128	€	129	□	130	,	131	∫
132	№	133		134		135		136		137	
138		139		140		141	□	142	□	143	□
144	□	145		146		147		148		149	
150		151		152		153		154		155	
156		157	□	158	□	159		160	_	161	
162		163		164		165		166		167	
168		169		170		171		172		173	
174		175		176		177		178		179	
180		181		182		183		184		185	
186		187		188		189		190		191	
192		193		194		195		196		197	
198		199		200		201		202		203	
204		205		206		207		208		209	
210		211		212		213		214		215	
216		217		218		219		220		221	
222		223		224		225		226		227	
228		229		230		231		232		233	
234		235		236		237		238		239	
240		241		242		243		244		245	
246		247		248		249		250		251	
252		253		254		255					

Tabella B.1: La tabella ASCII estesa approvata nel 1998.

Bibliografia

[1] ATO-IT Admin. ATO-IT. Banca Dati, 2001. http://www.di.unipi.it/∼ato-it/amigloss20.html.

[2] FolDoc Admin. Foldoc. Banca Dati, 2002. http://wombat.doc.ic.ac.uk/foldoc/index.html.

[3] GNU Site admin. The linux site. Web Directory, 2002.

[4] A. Aho, J. Hopcroft, and J. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.

[5] Alfred V. Aho, Ravi Sethi, , and Jeffrey D. Ullman. *Compilers: Principles, Techniques, and Tools*. Addison-Wesley, Reading, Mass., 1986.

[6] H. Alvestrand. Tags for the identification of languages. Technical Report RFC 1766, IETF (Internet Engineering Task Force), 1995.

[7] David A.Patterson and John L. Hennessey. *Computer organization & design, hardware/software interface*. Morgan Kaufmann Publishers, 1998.

[8] T. Berners-Lee, R. Fielding, and L. Masinter. Uniform resource identifiers (uri): Generic syntax and semantics. Technical report, W3C Consortium, 1997. Work in progress; see updates to RFC1738.

[9] Anne Brggemann-Klein. Regular expressions into finite automata. *Theoretical Computer Science*, 120:197–213, 1993. . Extended abstract in I. Simon, Hrsg., LATIN 1992, S. 97-98, Springer-Verlag, Berlin 1992.

[10] Anne Brggemann-Klein and Derick Wood. Deterministic regular languages. Technical Report Bericht 38, Universitt Freiburg, Institut fr Informatik, Oktober 1991.

[11] Stephen D. Burd and L.G. Meares. *Systems Architecture : Hardware and Software in Business Information Systems*. Boyd & Fraser Pub Co, 1994.

[12] European Commission. Open information interchange - commisione europea. Banca Dati, 2002. http://www2.echo.lu/oii/en/alpha.html.

[13] Ernesto Damiani. *Internet, guida pratica alla rete internazionale*. Tecniche nuove, 2000.

[14] Museo di Milano. Museo della scienza e della tecnica di milano on–line: Sezione informatica – sottosezione evoluzione dei sistemi di calcolo. Banca Dati, 2002. http://www.museoscienza.org/computer/calcolo/.

[15] Franco Fummi, Maria Giovanna Sami, and Cristina Silvano. *Progettazione Digitale*. McGraw-Hill, 2002.

[16] Steve Heath and Steve Halladay. *Microprocessor Architectures and Systems : Risc, Cisc and Dsp*. BUTTERWORTH-HEINEMANN, 1991.

[17] J. Hopcroft. *The design and Analysis of Computer Algorithms*. Addison-Wesley, 1 edition, 1974.

[18] Miles Ian, Rush Howard, and Turner Kevin. *I.T. Information Technology - Orizzonti ed implicazioni sociali delle nuove tecnologie*. Baskerville, 2000.

[19] B.W. Kerninghan and D.M. Ritchie. *Il Linguaggio C: seconda edizione*. Jackson Libri, 1997.

[20] Donald E. Knuth. *The TEXbook*. Addison-Wesley, 1984.

[21] Leslie Lamport. *LATEX: A Document Preparation System*. Addison-Wesley, 1986.

[22] Francesco Longo. il dizionario informatico. Banca Dati, 1997. http://www.dsi.unive.it/∼flongo/diz.

[23] Dino Mandrioli and Carlo Ghezzi. *Theoretical Foundations of Computer Science*. John Wiley & Sons, Inc., New York; Chichester; Brisbane; Toronto, 1987.

[24] Dennis P.Curtin, Kim Foley, Kunal Sen, and Cathleen Morin. *Informatica di base*. McGraw-Hill, 2002.

[25] Robert Sedgewick. *Algorithms*. Addison-Wesley, 2 edition, 1986.

[26] Keld Simonsen and et al. Official names for character sets. Technical Report IETF RFC 1766, IANA (Internet Assigned Numbers Authority), 1987. ftp://ftp.isi.edu/in-notes/iana/assignments/character-sets.

[27] Andrew S.Tanenbaum. *Computer Networks (terza ed.) - versione italiana: Reti di computer*. Prentice Hall, 2000.

[28] A. S. Tanenbaum. *Architettura del Computer*. Prentice Hall International - Jackson, 1999.

[29] Microsoft Web Team. Microsoft network - italy. Web Directory, 2002.

[30] Autori Vari. Pc webopaedia. Banca Dati. http://www.pcwebopaedia.com.

[31] Various. Code for the representation of names of languages. Technical Report ISO 639:1988 (E), ISO (International Organization for Standardization), Geneva, The Switzerland, 1988.

[32] Various. Information technology – universal multiple-octet coded character set (ucs)– part 1: Architecture and basic multilingual plane. Technical Report ISO/IEC 10646-1993 (E), (International Organization for Standardization), Geneva, The Switzerland, 1993. plus amendments AM 1 through AM 7.

[33] Various. *The Unicode Standard, Version 2.0*. Addison-Wesley Developers Press, Reading, Mass., 1996.

[34] Various. Codes for the representation of names of countries and their subdivisions – part 1: Country codes. Technical Report ISO 3166-1:1997 (E), (International Organization for Standardization), Geneva, The Switzerland, 1997.

[35] AcronymFinder WebMaster. Acronymfinder. Banca Dati, 2002. http://www.acronymfinder.com/.

[36] Geek Webmaster. Geek glossary. Banca Dati, 2002. http://www.geek.com/glossary/glossary_search.htm.

[37] Incoming Webmaster. Incoming glossary. Banca Dati, 2002. http://www.incoming.com/s2glossary.html.

[38] OneLook WebMaster. Onelook. Banca Dati, 2002. http://www.onelook.com/.

[39] WathIs WebMaster. Whatis. Banca Dati, 2002. http://whatis.com/.

[40] Ron White. *Il Computer come e' fatto e come funziona*. Mondadori Informatica, 1999.

[41] Eduard A. Yakubaitis. *Network Architectures for Distributed Computing*. ALLERTON PR, 1983.

[42] Luciana Zou. *L'informatica*. Tascabili Economici Newton, 1999.

Indice analitico