

Logica Matematica
per Ingegneria Informatica

Luca Motto Ros
Dipartimento di Matematica
Politecnico di Torino
luca.mottoros@polito.it

2 marzo 2008

Nota importante. Queste pagine contengono appunti personali del docente e sono messe a disposizione nel caso possano risultare utili a qualcuno. In nessun caso vanno considerate come programma ufficiale del corso e in nessun caso eventuali errori, inesattezze o difformità rispetto alle lezioni qui contenuti possono essere considerati quale fonte autorevole in sede d'esame.

Indice

Introduzione	v
0.1 Logica e informatica	vii
0.2 Notazione e nozioni di base	xi
1 Logica proposizionale	1
1.1 La sintassi	1
1.2 Semantica	7
1.3 Traduzione dal linguaggio naturale al linguaggio formale . . .	12
1.4 Dimostrazioni formali	15
1.5 Il Principio di Induzione	24
1.6 Forme normali	29
1.7 Risoluzione	39

Introduzione

La logica è lo studio del ragionamento umano e dei suoi principi. Nei linguaggi naturali (come l'italiano, l'inglese, il francese e così via) è possibile formulare vari tipi di affermazioni:

1. Il Monte Bianco è alto.
2. Il sole è freddo.

Chiaramente, la nostra intuizione (e le nostre conoscenze) ci permettono di asserire che la prima frase è vera mentre la seconda è falsa. Tuttavia, la verità o falsità di tali frasi non dipende da un “ragionamento”, ma dalle osservazioni fisiche del mondo che ci circonda. La logica non si occupa di stabilire la *verità* di certe affermazioni (questo è compito delle varie scienze fisiche e naturali), ma di *dedurre* la verità di una frase da quella di un'altra, come è mostrato nell'esempio seguente:

1. Gli uomini sono mortali e Socrate è un uomo.
2. Socrate è mortale.

In questo caso, la logica ci permette di osservare che se la prima frase (“Gli uomini sono mortali e Socrate è un uomo”) è vera allora anche l'ultima frase (“Socrate è mortale”) è vera. Tuttavia vi sono anche affermazioni indipendenti dall'osservazione del mondo esterno ma che risultano problematiche:

Questa frase è falsa¹.

Assumiamo che la frase sia vera: allora ciò che asserisce (cioè il fatto di essere falsa) deve essere vero e dunque la frase è falsa, contraddizione! Viceversa, se assumiamo che la frase sia falsa allora possiamo concludere con un ragionamento analogo al precedente che la frase è vera, è questa è nuovamente una contraddizione. Quindi la frase non può essere né vera né falsa!

¹Questa è una delle molteplici varianti del *paradosso del mentitore*.

Sembrerebbe dunque che il nostro modo di ragionare sia in realtà incoerente e di conseguenza sia anche inutile. Ma è evidente che le cose non stanno così: moltissimi progressi sono stati compiuti con la “forza della ragione”, e vi sono tantissime situazioni (come le dimostrazioni matematiche, per esempio) in cui il nostro ragionamento non è assolutamente problematica. Il problema allora deve risiedere nel linguaggio naturale che ci permette di esprimere questi paradossi. Tale possibilità è dovuta al fatto che i linguaggi naturali si sono evoluti nel tempo a seconda dell’uso e delle situazioni storiche, con regole grammaticali spesso mutevoli. Da questa osservazione è nata l’esigenza di utilizzare un linguaggio “artificiale” quando si voglia studiare astrattamente un ragionamento ed una dimostrazione. In particolare, si è cercato di introdurre un nuovo “alfabeto” (fatto di *simboli*), delle “regole grammaticali” (*sintassi*) con cui formare parole e frasi via via più complicate e delle “regole di deduzione” che permettano di determinare la verità di una frase a partire da altre ad essa collegate. Per esempio, sono stati introdotti svariati simboli che fungono da “congiunzioni” per collegare due frasi e formarne così una più complessa.

Simbolo	Significato
\wedge o $\&$	e (congiunzione)
\vee	oppure (inclusivo, corrispondente al “vel” latino)
\oplus	oppure (esclusivo, corrispondente all’“aut-aut” latino ed all’XOR informatico)
\rightarrow o \Rightarrow	se... allora...
\leftrightarrow o \Leftrightarrow	se e solo se
\downarrow	né... né (NOR informatico)

In questa lista sono stati inclusi solo i simboli e le “congiunzioni” più comuni, ma molte altre potrebbero essere aggiunte. L’unica accortezza, per evitare che anche il nostro linguaggio artificiale presenti ambiguità, è quella di evitare particelle che abbiano diversi significati nel linguaggio naturale (come ad esempio il “ma”: si osservi il suo diverso significato in “quella casa è rossa ma non è color mattone” e in “vorrei uscire ma piove”), oppure specificare quale dei possibili significati si debba assegnare al simbolo corrispondente (come nel caso di “oppure” che è stato tradotto con due simboli corrispondenti ai due possibili significati).

Dopodiché bisogna passare a introdurre nuovi simboli e regole che permettano di tradurre le proposizioni semplici (come “Socrate è un uomo” o “le rane sono anfibi”). Naturalmente vi sono diversi modi di fare ciò, e questa caratteristica distingue un “tipo” di logica da un’altra. Per esempio, se decidiamo di usare un simbolo diverso per ogni proposizione semplice (il simbolo A per “Socrate è un uomo”, il simbolo B per “le rane sono anfibi” e così via) si ottiene la *logica proposizionale*, di cui ci occuperemo nel primo capitolo. Chiaramente, questo genere di “traduzione”, pur sufficiente in

molti casi, è un po' grossolano. Un approccio un po' più raffinato è dato dalla *logica del prim'ordine* o *predicativa* di cui ci occuperemo nel secondo capitolo: in questo caso, le proposizioni semplici considerate prima verranno tradotte in formule del tipo $U(s)$ e $\forall x(R(x) \rightarrow A(x))$. Accanto a queste due, vi sono molti altri “tipi” di logica (come la logica del second'ordine) di cui non ci occuperemo in queste dispense.

Ciascuno di questi linguaggi artificiali è stato creato in modo da non essere in grado di esprimere proposizioni del tipo “questa frase è falsa”, così da evitare paradossi e ambiguità. Questo significa che il “potere espressivo” di questi linguaggi artificiali è di gran lunga minore di quello dei linguaggi naturali: per contro, si ottiene una maggior precisione nello stabilire quando una certa frase sia conseguenza di altre.

Inoltre doteremo ciascuna logica di un modo per “interpretare” le formule (la *semantica*) e di regole per dedurre nuove formule da altre. In ciascun caso dimostreremo che il corrispondente sistema di deduzione è *corretto* (ovvero che una formula derivata da altre secondo queste regole è vera se sono vere le formule di partenza) e *completo* (ovvero che vi sono abbastanza regole per derivare ogni affermazione vera).

0.1 Logica e informatica

Lo studio della logica è intimamente collegato con quello dell'informatica teorica. Da una parte la logica permette di avere un linguaggio e delle tecniche per studiare l'informatica teorica, dall'altra l'informatica fornisce un ambiente “concreto” per implementare la logica. Vedremo alla fine del secondo capitolo che diversi linguaggi di programmazione (tra cui il Prolog) sono nati proprio dall'idea di vedere un programma come una “dimostrazione” (nel senso logico del termine). Tali linguaggi sono di tipo *dichiarativo*, dunque consentono di ottenere un tipo di programmazione molto diverso da quelli dei linguaggi “classici” (C++, Fortran), che sono di tipo *imperativo*.

Naturalmente tutto ciò è anche collegato allo studio della così detta *Intelligenza Artificiale*. Una delle attività caratteristiche dell'intelligenza umana è proprio quella del linguaggio e del ragionamento: l'interazione tra logica e informatica ha permesso di ottenere diversi progressi in questa direzione, come l'implementazione di *dimostratori automatici* (Otter, Mace) o di linguaggi di programmazione che consentono ad un computer di interagire con l'utente per mezzo di un linguaggio molto flessibile e vicino a quello naturale.

Vi è infine un settore di studio che rappresenta una specie di “anello di congiunzione” tra logica e informatica: lo studio della complessità. L'idea è quella di classificare i problemi in base a quanto sono difficili da risolvere. In generale, chiameremo *problema di decisione* un qualunque problema

(formulato nel linguaggio naturale) che ammetta come uniche risposte “Sì” oppure “No”.

Esempi di problemi di decisione sono i seguenti:

1. Dato un numero naturale n , è vero che n è pari?
2. Dato un qualunque grafo finito², è possibile trovare un *cammino* che passi per tutti i nodi del grafo una e una sola volta?

In entrambi i casi, dato come input un numero naturale n (o, rispettivamente, un grafo finito) la risposta al problema posto potrà essere solo “Sì” oppure “No”. Per esempio, per risolvere il primo problema si può controllare se l’ultima cifra di n (espresso in base 10) è una cifra tra 0, 2, 4, 6 oppure 8: se ciò è vero si risponde “Sì” altrimenti si risponde “No”; è facile controllare che quella che abbiamo proposto è una soluzione rapida ed efficiente del problema corrispondente.

Come si è visto in questo esempio, la soluzione di un problema di decisione consiste semplicemente di un *algoritmo*. Un algoritmo può essere visto semplicemente come un programma in un certo linguaggio di programmazione prefissato (per esempio potremmo scegliere il C++ o il Fortran): un insieme *finito* di istruzioni che, dato l’input, fanno compiere diverse operazioni (i *passi* dell’algoritmo) e producono come output una delle due risposte possibili. Naturalmente, questa è solo una definizione “intuitiva” di cosa sia un algoritmo (nozione che dovrebbe essere comunque già nota): è possibile darne una definizione precisa specificando quale linguaggio di programmazione si voglia usare e, di conseguenza, quali siano i singoli passi dell’algoritmo. Normalmente, si introducono le *macchine di Turing* (il primo prototipo teorico di computer) e si definiscono *algoritmi* i programmi per queste macchine e *passi* le singole operazioni che tali macchine compiono. In queste dispense non daremo tale definizione formale perché esula dagli argomenti del corso. Tuttavia continueremo a parlare di “algoritmi”, “passi” e “velocità di esecuzione” in maniera informale (ma coerente).

Un algoritmo *termina sempre* se, dato un qualsiasi input, fornisce un output in un numero finito di passi. Dato un problema di decisione P , un algoritmo si dice *corretto* per P se per ogni possibile input, se ogni volta l’algoritmo termina e l’output ottenuto è “Sì” allora tale risposta al problema di decisione P è quella giusta. Viceversa, l’algoritmo si dice *completo* se, dato un qualunque input tale che il problema abbia risposta “Sì” in relazione ad esso, si ha che l’algoritmo termina con output “Sì” su tale input.

Un problema di decisione P si dice *semidecidibile* se esiste un algoritmo corretto e completo per esso, mentre si dice *decidibile* se esiste un tale algoritmo che termini sempre.

²Un grafo finito è un insieme finito di punti (detti *nod*i collegati tra di loro mediante *lati*. Per una definizione precisa si veda il secondo Capitolo.

In realtà si può osservare che la maggioranza degli algoritmi non è decidibile, ma tutti quelli che considereremo in queste dispense lo saranno. Per esempio, è chiaro che l'algoritmo proposto come soluzione al problema di decisione riguardante i numeri pari dimostra che esso è decidibile, ma anche il secondo problema è decidibile. Infatti basta considerare il seguente algoritmo. Dato una qualsiasi grafo finito, si considerano tutti i possibili ordinamenti dei suoi vertici e si controlla se ciascuno di vertice è collegato da un lato del grafo con quello successivo (rispetto all'ordinamento considerato). Se ciò avviene l'algoritmo termina e fornisce "Sì" come output, altrimenti passa ad esaminare l'ordinamento successivo (se esiste) oppure termina fornendo come output "No" (se non vi sono altri ordinamenti da controllare). È facile verificare che tale algoritmo termina sempre ed è corretto e completo per il problema sui grafi proposto.

La differenza tra i due algoritmi consiste nel fatto che il primo risulta essere molto "efficiente" (in termini di tempo) mentre il secondo no. Charamente è normale che il tempo (ovvero il numero di passi) impiegati da un algoritmo a fornire un output sia correlato alla *lunghezza* dell'input: più sarà lungo quest'ultimo, più tempo verrà impiegato dall'algoritmo per fermarsi. La lunghezza dell'input è sempre un numero naturale e può essere formalmente definita come la sua lunghezza in bit (qualora l'algoritmo venga visto come un programma di un computer). Spesso però si usano anche delle opportune nozioni di lunghezza che sono più strettamente collegate con il tipo di problema e con la sua presentazione. In genere, questo migliora la comprensione dell'algoritmo (e del problema) e non costituisce un ostacolo nel determinare la complessità del problema stesso. Infatti, affinché le definizioni sequenti siano ben date, è necessario (e sufficiente) che la nozione di lunghezza utilizzata sia *polinomiale* rispetto a quella "ufficiale" (lunghezza in bit) ovvero tale che esista un polinomio $P(x)$ tale che per ogni possibile input, se la sua lunghezza rispetto alla nozione prescelta è n allora la sua lunghezza in bit è minore di $P(n)$.

Consideriamo un algoritmo che termini sempre. Diremo che l'algoritmo *lavora in tempo polinomiale* se esiste un numero naturale k tale che per ogni possibile input, se la sua lunghezza è $n > 1$ l'algoritmo termina in meno di n^k passi. Viceversa, l'algoritmo *lavora in tempo non polinomiale* se per ogni k ed ogni n sufficientemente grande esiste un input di lunghezza n tale che l'algoritmo termina in più di n^k passi. Gli algoritmi che lavorano in tempo polinomiale sono algoritmi "veloci", che possono essere implementati su un computer ottenendo buone prestazioni (almeno per k piccoli). I secondi invece sono lenti, nel senso che se vengono implementati su un computer, per quanto questo possa essere veloce e potente non sarà mai in grado di fornire un output in un tempo ragionevole (rispetto alla lunghezza dell'input).

Il primo algoritmo proposto (quello relativo ai numeri pari) è un algoritmo che lavora in tempo polinomiale: dato un qualunque input, l'algoritmo richiede solo 5 passi (corrispondenti a confrontare l'ultima cifra dell'input

con i numeri pari minori di 10) per fornire l'output corretto. Viceversa, l'algoritmo riguardante i grafi è non polinomiale. Se assumiamo come nozione di lunghezza di un input (cioè di un grafo) il numero n dei suoi vertici, bisognerà in generale (per esempio quando l'output da fornire sia "No") controllare $n!$ ordinamenti (ovvero tutti i possibili modi per ordinare n oggetti), ed è chiaro che per ogni k si ha $n^k < n!$ per n sufficientemente grande (ovvero l'algoritmo lavora in tempo non polinomiale). Se si implementasse tale algoritmo sul computer più potente del mondo e si fornisse come input un grafo con 100 vertici (dunque un grafo che di per sé è relativamente "piccolo") che non ammetta un cammino che tocchi i tutti i suoi vertici esattamente una volta, il tempo richiesto per computare l'output dell'algoritmo sarebbe di gran lunga maggiore rispetto alla vita della Terra. È chiaro quindi che gli algoritmi che non lavorano in tempo polinomiale non sono "efficienti".

Questa nozione di "velocità di terminazione" di un algoritmo permette di distinguere (tra tutti i problemi decidibili) quelli che lo sono "realmente" (cioè ammettono un algoritmo che li risolve in maniera "efficiente") e quelli che lo sono solo "in teoria". Chiameremo algoritmo *deterministico* un algoritmo in cui ogni passo è completamente determinato (come lo sono in generale i programmi per computer attuali). Un problema apparterrà alla classe **P** se ammette come soluzione un algoritmo deterministico che lavora in tempo polinomiale. Dunque la classe **P** contiene tutti i problemi "facilmente risolvibili" (come ad esempio il problema relativo ai numeri pari). Accanto alla nozione di algoritmo deterministico, poniamo però la nozione di algoritmo *non deterministico*, cioè di un algoritmo in cui talora il passo successivo possa essere scelto arbitrariamente tra un numero *finito* di possibilità (cioè "scommettendo" su una di tali possibilità). In particolare, ogni algoritmo deterministico è anche un algoritmo non deterministico. Fissato un input che dia risposta positiva, ad ogni "scommessa" corrisponde una certa computazione dell'output "Sì" con un corrispondente numero di passi (eventualmente dipendente dalla lunghezza dell'input): definiamo *velocità (globale) di terminazione* delle risposte positive fornite dall'algoritmo come il minimo tra tali numeri. Dunque, un algoritmo non deterministico darà risposta "Sì" in *tempo polinomiale* (ovvero *lavora in tempo polinomiale sulle risposte positive*) se la velocità globale di terminazione su un input che ammetta una tale risposta dipende in maniera polinomiale dalla lunghezza $n > 1$ dell'input, ovvero se esiste un numero naturale k tale che la velocità globale di terminazione sugli input che ammettono risposta positiva è inferiore a n^k . Chiaramente ogni algoritmo non deterministico può essere convertito in un algoritmo deterministico semplicemente imponendo di scorrere in successione tutte le possibilità anziché "scommettere" su una in particolare: tuttavia, un algoritmo non deterministico che fornisce risposte "positive" in tempo polinomiale può dar luogo ad un algoritmo deterministico che lavora in tempo non polinomiale. In ogni caso, un problema apparterrà alla classe **NP** se ammette un algoritmo *nondeterministico* per

esso che lavora in tempo polinomiale sulle risposte positive. Per esempio, ritorniamo al problema relativo ai grafi e si consideri il seguente algoritmo:

- scegliere un ordinamento dei nodi del grafo;
- controllare che ogni nodo sia collegato al successivo da un lato del grafo.

Tale algoritmo è non deterministico perché al primo passo bisogna scegliere (ovvero “scommettere” su) un possibile ordinamento dei vertici (tali ordinamenti sono $n!$ se il grafo ha n vertici, quindi sono sempre in numero finito). Chiaramente questo algoritmo lavora in tempo polinomiale sulle risposte positive: se siamo fortunati ed al primo passo scegliamo l’ordinamento giusto, basta controllare che questo corrisponda ad un cammino per fornire l’output “Sì”. Dunque basta compiere una volta il primo passo e poi controllare che vi siano $n - 1$ lati (se il grafo ha n vertici) che colleghino tali vertici nel giusto ordine. Questo prova che il problema di decisione proposto è in **NP**. Tuttavia, se da questo algoritmo si passa alla sua “versione deterministica” si riottiene l’algoritmo proposto all’inizio che, come abbiamo visto, non lavora in tempo polinomiale.

Poiché gli algoritmi deterministici sono anche non deterministici, è facile verificare che $\mathbf{P} \subseteq \mathbf{NP}$. Il problema di stabilire se valga anche l’inclusione inversa (cioè se $\mathbf{P} = \mathbf{NP}$) è uno dei problemi aperti più importanti e difficili in matematica e informatica teorica (il Clay Institute of Mathematics ha posto una “taglia” di un milione di dollari sulla soluzione di tale problema). Infatti, il semplice fatto che non siamo in grado di trovare un algoritmo deterministico che lavori in tempo polinomiale e risolva un certo problema in **NP** non significa che un tale algoritmo non esista! Tuttavia esistono alcuni problemi P (come il problema della soddisfacibilità per la logica proposizionale o il problema dei grafi proposto prima) che sono **NP**-completi, ovvero stanno in **NP** e sono tali che se essi appartengono a \mathbf{P} allora $\mathbf{P} = \mathbf{NP}$. Questi risultati vengono ottenuti mostrando come sia possibile, dato un qualunque problema P' in **NP**, convertire un qualunque algoritmo deterministico per P che lavori in tempo polinomiale in un corrispondente algoritmo deterministico per P' che lavori anch’esso in tempo polinomiale. Quindi per risolvere il problema $\mathbf{P} = \mathbf{NP}$ ci si può concentrare solamente sui problemi **NP**-completi: se si trovasse un algoritmo deterministico “efficiente” che li risolva si otterrebbe $\mathbf{P} = \mathbf{NP}$, mentre se si dimostrasse che tale algoritmo non esiste si avrebbe automaticamente che $\mathbf{P} \neq \mathbf{NP}$.

0.2 Notazione e nozioni di base

Considereremo innanzitutto la nozione di *insieme* come una nozione intuitiva. Un insieme è semplicemente una qualunque collezione di oggetti (simboli, numeri, oggetti fisici e così via) senza una ordine particolare e senza

ripetizioni: dunque le seguenti collezioni

$$\{0, 1\} \quad \{0, 1, 1\} \quad \{1, 0\}$$

saranno tutte considerate come lo *stesso insieme*. La notazione riguardante gli insiemi che useremo è abbastanza standard.

- $x \in A$ significa che x è un elemento di A (*appartiene ad A*);
- $x \notin A$ significa che x non appartiene ad A ;
- \emptyset è l'(unico) insieme vuoto (cioè non contiene nessun elemento);
- $A \subseteq B$ significa che ogni elemento che sta in A appartiene anche a B , ovvero che A è un *sottoinsieme* di B ;
- $A = B$ significa che A e B sono lo stesso insieme (contengono esattamente gli stessi elementi);
- $A \neq B$ significa che A e B sono diversi;
- $A \subset B$ o $A \subsetneq B$ significa che $A \subseteq B$ ma $A \neq B$;
- $A \cup B$ denota l'*unione* degli insiemi A e B ;
- $A \cap B$ denota l'*intersezione* degli insiemi A e B ;
- $A \times B$ denota il *prodotto cartesiano* degli insiemi A e B ;
- $A \setminus B$ denota la differenza tra A e B .

L'unione $A \cup B$ di A e B è la collezione di tutti gli elementi che appartengono ad A oppure a B (compresi quelli che stanno sia in A che in B), mentre l'intersezione $A \cap B$ è la collezione di tutti gli elementi che stanno sia in A che in B . Il prodotto cartesiano $A \times B$ di A e B è l'insieme di tutte le *coppie ordinate* (a, b) tali che $a \in A$ e $b \in B$. Scriveremo A^2 al posto di $A \times A$, mentre A^n (con $n > 2$) denoterà sia il prodotto cartesiano $A^{n-1} \times A$, sia l'insieme delle n -uple ordinate (a_1, \dots, a_n) con ogni $a_i \in A$ (tali insiemi sono formalmente diversi, ma verranno identificati per via della naturale biiezione che esiste tra di loro). Per semplicità di notazione considereremo anche $A^1 = A$. Infine la differenza $A \setminus B$ è la collezione di tutti gli elementi che stanno in A ma non in B .

Esempio. Siano $A = \{0, 1, 2\}$ e $B = \{1, 3\}$. Allora:

- $A \cup B = \{0, 1, 2, 3\}$
- $A \cap B = \{1\}$
- $A \times B = \{(0, 1), (0, 3), (1, 1), (1, 3), (2, 1), (2, 3)\}$

$$- A \setminus B = \{0, 2\}$$

Se $A \subsetneq B$ e $A \neq \emptyset$ diciamo che A è un *sottoinsieme proprio* di B (dunque i sottoinsiemi *impropri* di A sono esattamente l'insieme vuoto \emptyset e A stesso). L'*insieme potenza* di $\mathcal{P}(A)$ è la collezione di tutti i sottoinsiemi (propri ed impropri) di A .

Definizione 1. Sia A un insieme. Una relazione n -aria R su A (con $n \geq 1$) è semplicemente un qualunque sottoinsieme di A^n , cioè una collezione di n -uple ordinate (a_1, \dots, a_n) tali che $a_i \in A$ per ogni $1 \leq i \leq n$.

Il numero naturale n viene detto *arietà della relazione* R e se $n = 1$ (rispettivamente, $n = 2$ o $n = 3$) diremo che R è una relazione unaria (rispettivamente, binaria o ternaria).

Osserviamo che le relazioni unarie sono in realtà semplici sottoinsiemi di A (poiché $A^1 = A$), e per questa particolarità verranno anche dette *proprietà* o *predicati*. Infatti, dato un insieme A , la collezione di tutti gli elementi di A che soddisfano una certa proprietà P (nel senso comune del termine) è chiaramente un sottoinsieme di A , dunque una relazione unaria $R_P \subseteq A$. Viceversa, data una relazione unaria $R \subseteq A$ è possibile definire la proprietà P_R come “un elemento ha la proprietà P_R se e solo se appartiene ad R ”. Dunque proprietà e relazioni unarie possono essere identificate, e diremo che due proprietà sono *equivalenti* se danno luogo alla stessa relazione unaria (per esempio, la proprietà di “essere un numero pari” e quella di “terminare con 0, 2, 4, 6 o 8” sono proprietà equivalenti sui numeri naturali).

Anche le relazioni binarie occupano un posto particolare (specialmente in matematica). Diciamo che una relazione binaria R su A è

- *transitiva* se per ogni $a, b, c \in A$, se $(a, b) \in R$ e $(b, c) \in R$ allora $(a, c) \in R$;
- *riflessiva* se $(a, a) \in R$ per ogni $a \in A$;
- *simmetrica* se per ogni $a, b \in A$ tali che $(a, b) \in R$ si ha anche che $(b, a) \in R$;
- *antisimmetrica* se per ogni $a, b \in A$ tali che $(a, b) \in R$ e $(b, a) \in R$ si ha che $a = b$;
- *connessa* se per ogni $a, b \in A$ si ha che o $(a, b) \in R$ oppure $(b, a) \in R$.

Diremo che R è un *ordine parziale* se è riflessiva, transitiva e antisimmetrica, mentre è un *ordine* se è un ordine parziale che è anche connesso. Infine, diremo che R è una *relazione di equivalenza* se è riflessiva, transitiva e simmetrica. Se R è una relazione di equivalenza, dato un elemento $a \in A$

si può definire la sua *classe di equivalenza (rispetto a R)* $[a]_R$ come la collezione di tutti i $b \in A$ tali che $(a, b) \in R$. Si osservi che le varie classi di equivalenza sono sempre a due a due disgiunte (per la transitività di R) e che la loro unione dà l'intero insieme di partenza A (cioè le classi di equivalenza di una qualunque relazione di equivalenza formano una *partizione* di A). Viceversa, data una qualunque partizione di A (cioè una qualunque collezione $\mathcal{A} \subseteq \mathcal{P}(A)$ di sottoinsiemi di A tale che l'unione dei suoi elementi è uguale ad A e i suoi elementi sono tutti a due a due disgiunti) si può associare ad essa una relazione di equivalenza $R_{\mathcal{A}}$ definita ponendo $(a, b) \in R_{\mathcal{A}}$ se e solo se esiste un elemento di \mathcal{A} che contiene sia a che b (è facile verificare che $R_{\mathcal{A}}$ è veramente una relazione di equivalenza). Dunque partizioni di un insieme A e relazioni di equivalenza binarie su di esso possono essere identificate. Infine, il *quoziente di A rispetto a R* è semplicemente la collezione di tutte le classi di equivalenza rispetto a R .

Data una qualunque relazione binaria R su A possiamo inoltre definire la sua *relazione inversa* R^{-1} come l'insieme di tutte le coppie (a, b) tali che $(b, a) \in R$. È facile verificare che se R è un ordine parziale (rispettivamente, un ordine o una relazione di equivalenza) anche R^{-1} è un ordine parziale (rispettivamente, un ordine o una relazione di equivalenza).

Una *funzione* $f : A \rightarrow B$ è semplicemente un sottoinsieme f di $A \times B$ tale che per ogni $a \in A$ esiste esattamente un elemento $b \in B$ tale che $(a, b) \in f$. In particolare denoteremo b come $f(a)$. Dunque se $A = B^n$ (con $n \geq 1$) una funzione $f : A \rightarrow B$ è semplicemente una relazione $n + 1$ -aria su B con delle proprietà particolari. Inoltre, in questo caso, diremo che f è una *funzione n -aria da B in B* e che n è l'*arietà* di f . L'insieme A viene detto *dominio* di f e viene denotato a volte con $\text{dom}(f)$, mentre la collezione di tutti gli elementi di B per cui esiste una $a \in A$ tale che $f(a) = b$ verrà detto *range* (o *codominio*) di f e verrà denotato con $\text{range}(f)$. Una funzione $f : A \rightarrow B$ verrà detta *iniettiva* se per ogni $a, a' \in A$ tali che $a \neq a'$ si ha che $f(a) \neq f(a')$, *suriettiva* se $\text{range}(f) = B$ e *bijettiva* se è sia iniettiva che suriettiva. Inoltre una funzione iniettiva (rispettivamente, suriettiva o biiettiva) verrà anche detta *iniezione* (rispettivamente, *suriezione* o *biezione*).

La funzione $id = id_A : A \rightarrow A$ tale che $id(a) = a$ per ogni $a \in A$ verrà detta *identità (su A)*. Se $f : A \rightarrow B$ è una funzione e $A' \subseteq A$ denoteremo con $f(A')$ l'insieme dei $b \in B$ tali che esiste $a \in A'$ per cui $f(a) = b$. Viceversa, dato $B' \subseteq B$ denoteremo con $f^{-1}(B')$ l'insieme degli $a \in A$ tali che $f(a) \in B'$. Date due funzioni $f : A \rightarrow B$ e $g : B \rightarrow C$ possiamo definire la loro *composizione* $g \circ f : A \rightarrow C$ ponendo $g \circ f(a) = g(f(a))$ per ogni $a \in A$.

Dalla definizione data, è chiaro che una funzione $f : A \rightarrow B$ coincide sempre con il suo *grafo* $\text{graph}(f)$, ovvero con la collezione delle coppie ordinate $(a, b) \in A \times B$ tali che $f(a) = b$. Dunque per definire una funzione

si possono usare due modi: fornire una regola per calcolare $f(a)$ a partire dall'input $a \in A$, oppure descrivere il suo grafo. Per esempio $f(x) = 2x^2 + 3$ definisce una funzione dai numeri reali nei numeri reali mediante la seguente regola: dato un numero reale x lo si eleva al quadrato, lo si moltiplica per 2 e poi si somma 3. Un altro esempio è dato dalla funzione $g : \{0, 1\} \rightarrow \{0, 1\}$ definita ponendo $g(x) = 1 - x$. Ma la stessa funzione può essere anche definita semplicemente definendone il grafo $\text{graph}(g) = \{(0, 1), (1, 0)\}$ oppure associando ad ogni elemento del dominio il corrispondente valore, ovvero fornendo un elenco del tipo $g(0) = 1$ e $g(1) = 0$ (quest'ultima definizione si può chiaramente dare solo quando il dominio della funzione è un insieme finito, poiché in caso contrario si potrebbe avere un elenco infinito).

Data una funzione $f : A \rightarrow B$ si può anche considerare il suo *grafo inverso*, ovvero l'insieme delle coppie $(b, a) \in B \times A$ tali che $(a, b) \in \text{graph}(f)$ (ovvero tali che $f(a) = b$). È chiaro che il grafo inverso di una funzione può non essere una funzione, ma se ciò accade si ottiene la *funzione inversa* $f^{-1} : \text{range}(f) \rightarrow A$ di f (ed è facile verificare che ciò accade esattamente quando f è iniettiva). Inoltre, poiché $f^{-1}(b) = a$ se e solo se $f(a) = b$, si ha che $f^{-1}(f(a)) = a$ per ogni $a \in A$, ovvero che $f^{-1} \circ f$ è l'identità su A (analogamente $f \circ f^{-1}$ è l'identità su $\text{range}(f)$).

Capitolo 1

Logica proposizionale

Come abbiamo visto nell'introduzione, una logica è costituita da un linguaggio, delle regole grammaticali (sintassi) per costruire formule via via più complesse, delle regole per interpretare i simboli (semantica) e delle regole per derivare delle formule da altre formule. In questo Capitolo ci occuperemo della logica che va sotto il nome di *logica proposizionale*: si tratta di una logica molto semplice e con un "potere espressivo" molto limitato, ma già a questo livello di analisi sorgono questioni molto importanti (per esempio, il problema della soddisfacibilità per la logica proposizionale è **NP**-completo). Inoltre la logica proposizionale servirà come "base" per costruire la logica del prim'ordine di cui ci occuperemo nel capitolo seguente.

1.1 La sintassi

Un linguaggio per la logica proposizionale è un insieme di *simboli* (possibilmente infinito) del tipo

$$\mathcal{L} = \{\neg, \wedge, (,), A_0, A_1, \dots\}.$$

I simboli \neg e \wedge vengono dette *costanti logiche* (o *simboli fissi* o *connettivi*) poiché la loro interpretazione (semantica) verrà fissata una volta per tutte, le parentesi (e) vengono detti *simboli tipografici* (poiché verranno usati solo per facilitare la lettura delle formule, ma in teoria potrebbero essere eliminati) , mentre i restanti simboli A_0, A_1, \dots vengono dette *lettere proposizionali* o *formule atomiche*. I linguaggi della logica proposizionale sono dunque distinti dalla scelta delle lettere proposizionali che vi inseriamo¹. Poiché le costanti logiche e i simboli tipografici sono comuni a tutti i

¹In realtà si potrebbe presentare la logica proposizionale definendo un *unico* linguaggio proposizionale con una quantità infinita (ma numerabile) di lettere proposizionali. Si tratta di un approccio equivalente e non sarà difficile capire come modificare le definizioni e i teoremi seguenti per adattarli al differente contesto.

linguaggi proposizionali, spesso identificheremo il linguaggio \mathcal{L} con l'insieme $\{A_0, A_1, \dots\}$ delle sue lettere proposizionali.

Come si può osservare, dei simboli per le “congiunzioni” proposti nell'introduzione ne abbiamo inseriti nella nostra lista delle costanti logiche soltanto due (\neg e \wedge): non bisogna però pensare che questo ponga una limitazione al potere espressivo della logica proposizionale. Infatti “definiremo” gli altri simboli sulla base di questi due in maniera che la loro interpretazione (semantica) sarà esattamente quella desiderata. La ragione per effettuare questa scelta (ovvero ridurre il numero delle costanti logiche ammesse) risiede nel fatto che, come vedremo, per dimostrare teoremi riguardanti le formule della logica proposizionale sarà conveniente utilizzare meno costanti logiche possibili.

Definizione 2. *L'insieme delle formule (proposizionali) (o proposizioni) di un certo linguaggio \mathcal{L} è definito induttivamente nella seguente maniera:*

1. ogni formula atomica (lettera proposizionale) è una formula;
2. se F è una formula anche $\neg F$ lo è, e se F e G sono formule anche $(F \wedge G)$ lo sono.

Dunque ogni stringa finita di simboli di un linguaggio \mathcal{L} costruita a partire dalle formule atomiche utilizzando la clausola 2 è una formula.

Data una stringa finita di simboli F possiamo definire anche il suo linguaggio $\mathcal{L}(F)$ come l'insieme delle lettere proposizionali (ovvero di tutti i simboli che non sono costanti logiche o simboli tipografici) che compaiono in essa. In questo caso F verrà detta *formula* se è una formula del linguaggio $\mathcal{L}(F)$. Si osservi inoltre che, poiché una formula di qualunque linguaggio proposizionale è sempre una stringa finita di simboli, si ha che il linguaggio di ogni formula è sempre finito.

Osservazione. Cos'è una definizione induttiva?

Sia dato una classe di oggetti \mathcal{U} e si voglia definire induttivamente un suo sottoinsieme C . Tale definizione consta di una coppia il cui primo elemento è un insieme $B \subseteq \mathcal{U}$ ed il secondo è un insieme \mathcal{F} di funzioni (di fissata arietà) che mandano (n -uple di) elementi di \mathcal{U} in elementi di \mathcal{U} , unita alle seguenti clausole:

1. ogni elemento di B è un elemento di C ;
2. se a_0, \dots, a_{n-1} è una n -upla di elementi di C e f è una funzione n -aria appartenente ad \mathcal{F} allora $f(a_0, \dots, a_{n-1})$ è un elemento di C ;
3. nessun altro elemento appartiene a C .

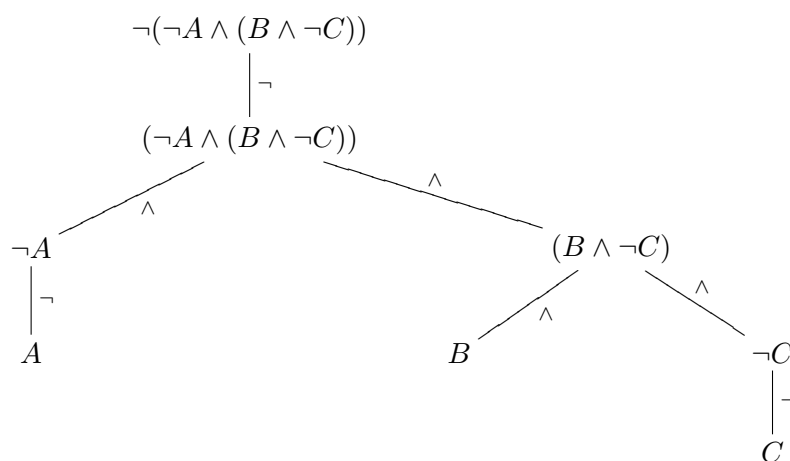
Dunque C è costituito dagli elementi di \mathcal{U} che si possono ottenere dagli elementi di B applicandovi un numero finito di volte (eventualmente nessuna) le funzioni di \mathcal{F} . Equivalentemente, C è il più piccolo insieme contenente B e chiuso rispetto alle funzioni di \mathcal{F} ².

²Un insieme X si dice *chiuso rispetto alla funzione n -aria f* se per ogni n -upla x_0, \dots, x_{n-1} di elementi di X , anche $f(x_0, \dots, x_{n-1})$ è un elemento di X .

Nel nostro caso si considera l'insieme \mathcal{S} di tutte le stringhe finite di elementi del linguaggio \mathcal{L} , e si definisce l'insieme delle formule ponendo $B = \{\text{formule atomiche}\}$ e $F = \{f_{\neg}, f_{\wedge}\}$ dove f_{\neg} ed f_{\wedge} sono le funzioni, rispettivamente, unaria e binaria da \mathcal{S} in \mathcal{S} tali che $f_{\neg}(F) = \neg F$ e $f_{\wedge}(F, G) = (F \wedge G)$.

Ad ogni elemento di un insieme definito induttivamente si può associare un *albero di costruzione*. Nel caso delle formule proposizionali si ottiene l'*albero sintattico*, che è un albero binario e finito. In cima (la *radice* dell'albero) si pone la stringa di simboli (di un linguaggio proposizionale) F che si vuole analizzare. Se la stringa è una formula atomica (ovvero contiene solo una lettera proposizionale) la costruzione dell'albero è terminata. In caso contrario potremmo pensare che F sia stata ottenuta applicando la clausola 2 ad altre formule e vi sono due casi: se F è del tipo $\neg G$ si appende un unico ramo (indicizzato con \neg) che termina in un nodo in cui si pone G , mentre se F è del tipo $(G \wedge H)$ si appendono dei rami (indicizzati con \wedge) che terminino nei nodi G e H . Se la stringa F non ricade in uno dei casi che abbiamo descritto significa che F non è una formula proposizionale e non possiamo completarne l'albero di costruzione. Se invece ciò non accade, si considerano i nuovi nodi che abbiamo introdotto e si applica nuovamente la procedura appena descritta. È chiaro che ogni stringa F o non è una formula (e questo avviene se e solo se non possiamo completarne l'albero di costruzione), oppure la costruzione terminerà e avremo un albero che (letto dal basso verso l'alto) descrive come la formula F possa essere costruita a partire da certe formule atomiche (quelle che stanno nei *nodi terminali*, ovvero nei nodi a cui non sono appesi altri rami).

Esempio. Sia data la formula $F = \neg(\neg A \wedge (B \wedge \neg C))$. Il suo albero di costruzione è:



Se si riesce a completare l'albero di costruzione di F (ovvero se F è una

formula proposizionale), ogni formula che compaia in un nodo del suo albero di costruzione (compresa F stessa) si dice *sottoformula di F* . Una sottoformula di F si dice *propria* se è diversa da F . La definizione di sottoformula si può dare anche senza ricorrere agli alberi di costruzione nella maniera seguente.

Definizione 3. *Ogni formula è una sottoformula di se stessa. Ogni sottoformula di una formula F è anche una sottoformula di $\neg F$. Ogni sottoformula di F o G è anche una sottoformula di $(F \wedge G)$.*

Si osservi che ogni sottoformula di F è sempre una sottostringa di F , ma non ogni sottostringa di F è una sottoformula di F .

Ad ogni insieme C definito induttivamente si possono anche associare una o più *graduazioni*, cioè delle funzioni $d : C \rightarrow \mathbb{N}$ tali che per ogni elemento base b (cioè, riferendosi all'osservazione precedente, appartenente a B) $d(b) = \min\{n : n \in \text{range}(d)\}$ (generalmente $d(b) = 0$), e ad ogni elemento $c \in C \setminus B$ la funzione d associa un numero (strettamente) maggiore di quelli che associa agli elementi da cui c è stato ottenuto. Il valore della graduazione d su un dato elemento $c \in C$ si dice *grado* dell'elemento c . Nel caso proposizionale sono graduazioni le seguenti funzioni: l'altezza dell'albero sintattico³, la lunghezza della stringa e il numero di occorrenze di connettivi. Le graduazioni sono importanti perché, come vedremo, consentiranno di dimostrare vari teoremi sulle formule proposizionali *per induzione sulla complessità della formula*, ovvero per induzione sul *grado* della formula.

In realtà, quando si devono scrivere formule proposizionali, si usano generalmente anche tutti gli connettivi (come $\vee, \rightarrow, \leftrightarrow$) per facilitare la comprensione della formula stessa. Formalmente essi devono essere visti come abbreviazioni di formule contenenti solo \neg e \wedge secondo la seguente convenzione:

1. $F \vee G$ è un'abbreviazione per $\neg(\neg F \wedge \neg G)$;
2. $F \rightarrow G$ è un'abbreviazione per $\neg F \vee G$;
3. $F \leftrightarrow G$ è un'abbreviazione per $((F \rightarrow G) \wedge (G \rightarrow F))$;
4. $F \oplus G$ è un'abbreviazione per $((F \vee G) \wedge \neg(F \wedge G))$;
5. $F \downarrow G$ è un'abbreviazione per $(\neg F \wedge \neg G)$.

³Nel seguito il grado di una proposizione sarà, convenzionalmente, l'altezza di tale albero diminuita di 1. Bisogna tener conto però che, siccome abbiamo scelto come primitivi i connettivi \neg e \wedge , per calcolare il grado di una formula proposizionale bisogna costruirne l'albero utilizzando solo tali connettivi (ovvero riscrivere la formula usando le definizioni di \vee, \rightarrow e \leftrightarrow e poi costruirne l'albero).

Vedremo che tali definizioni sono “ben date” nel senso che, per esempio, la tavola di verità di $F \vee G$ (ovvero di $\neg(\neg F \wedge \neg G)$) corrisponderà al significato che volevamo dare al simbolo \vee (vedi Introduzione).

Data una formula F non atomica (contenente eventualmente anche i nuovi simboli \vee , \rightarrow e così via) chiameremo *connettivo più esterno* il connettivo \bullet tale che F è del tipo $G \bullet H$ (se \bullet è uno tra \wedge , \vee , \rightarrow , \leftrightarrow , \oplus o \downarrow) oppure F è del tipo $\bullet G$ (se \bullet è \neg). F verrà detta *coniunzione* (rispettivamente, *negazione*, *disgiunzione*, *implicazione* o *biimplicazione*) quando il suo connettivo più esterno è \wedge (rispettivamente, \neg , \vee , \rightarrow o \leftrightarrow).

Un'altra convenzione (oltre alle abbreviazioni introdotte precedentemente) che aiuta la comprensione immediata di una formula è quella riguardante l'utilizzo di parentesi. Chiaramente le parentesi sono state introdotte per evitare ogni possibile ambiguità di lettura, e se si volessero implementare le formule su d un computer il numero di parentesi non costituirebbe certo un problema. Ma se si provano a scrivere proposizioni via via più complicate utilizzando la definizione formale di “formula”, ci si renderà conto ben presto che le parentesi aumenteranno molto velocemente in numero e diventerà difficile per una persona umana tener conto durante la lettura della formula di quante parentesi siano state “aperte” e quante siano state già “chiuse”. Per questa ragione adotteremo le seguenti convenzioni.

Convenzioni sulle parentesi.

1. Si possono utilizzare anche parentesi quadre o graffe al posto di quelle tonde;
2. si possono omettere le parentesi che racchiudono l'intera formula;
3. ai connettivi viene convenzionalmente data la seguente priorità:

$$\neg$$

$$\wedge, \vee$$

$$\rightarrow$$

$$\leftrightarrow$$

intendendo con questo che i connettivi posti più in alto “legano” più strettamente di quelli in basso. Seguendo questa convenzione, si può omettere una coppia di parentesi purché tale operazione non alteri la tavola di verità (si veda la Sezione successiva) della formula in questione (si tenga presente, ad esempio, che $\neg(A \wedge B)$ e $\neg A \wedge B$ hanno tavole di verità diverse e quindi tali parentesi *non* si possono omettere);

4. per l'associatività di \wedge (vedi più avanti), le scritte $(A \wedge B) \wedge C$, $A \wedge (B \wedge C)$ e $A \wedge B \wedge C$ saranno considerate tutte la stessa formula (poiché le prime due hanno la stessa tavola di verità). Analogamente per il \vee .

Esercizi.

1. Dopo aver sostituito i connettivi diversi da \wedge e \neg con le corrispondenti formule, verificare quali delle seguenti stringhe sono formule proposizionali e quali no, costruire l'albero sintattico corrispondente e spiegare dove eventualmente la costruzione fallisce (e per quale ragione), determinare il grado (= altezza dell'albero - 1) della formula:

$$\neg\neg A$$

$$(A \rightarrow (B \vee \neg C))$$

$$\neg\neg(A \rightarrow B)$$

$$(((A \rightarrow B) \wedge A) \rightarrow B)$$

$$(\neg A \wedge B) \vee C$$

$$((\neg A \wedge B) \vee C)$$

2. Eliminare le parentesi non necessarie applicando le convenzioni:

$$(A \wedge (\neg B \rightarrow \neg A))$$

$$((\neg\neg\neg A \vee (A \wedge B)) \rightarrow (\neg A \wedge \neg B))$$

3. Indicare i connettivi principali delle seguenti stringhe. Reintrodurre poi le parentesi e sostituire i connettivi diversi da \wedge e \neg con la corrispondente formula, in modo da ottenere proposizioni (nel senso formale del termine). Costruirne poi l'albero sintattico ed elencare sottoformule e sottoformule proprie:

$$(\neg A \wedge B) \vee C$$

$$A \rightarrow B \vee \neg C$$

$$(A \rightarrow B) \wedge A \rightarrow B$$

$$A \rightarrow (B \wedge A \rightarrow B)$$

$$A \vee (B \wedge C) \rightarrow \neg A$$

$$(A \wedge B \wedge C) \vee \neg C$$

1.2 Semantica

Passiamo ora a considerare la semantica, ovvero ad associare delle *tavole di verità* alle costanti logiche. Si osservi che la logica proposizionale non si occupa di stabilire la verità o meno di proposizioni semplici (ovvero delle lettere proposizionali), ma semplicemente di dedurre la verità o falsità di una proposizione complessa a partire dalla verità o meno delle sue componenti. In quanto segue i *valori di verità* associati ad una formula saranno 0 per *falso* e 1 per *vero*.

Tavole di verità

Ad ogni connettivo (ovvero a \neg e \wedge) corrisponde una tavola di verità che ne determina *univocamente* la semantica.

A	$\neg A$
0	1
1	0

A	B	$A \wedge B$
0	0	0
0	1	0
1	0	0
1	1	1

Questo significa che, per esempio, il valore di verità della formula $\neg A$ è 1 (vero) se il valore di verità di A è 0 (falso) e viceversa. Si osservi dunque che la negazione cambia il valore di verità della formula di partenza e dunque corrisponde esattamente alla negazione dei linguaggi naturali.

Utilizzando queste due tavole, è possibile costruire la tavola di verità di qualunque formula proposizionale F calcolando via via i valori di verità di tutte le sue sottoformule (cioè “seguendo” il suo albero di costruzione). In particolare, si considerano tutte le possibile combinazioni di valori di verità da assegnare alle formule atomiche in $\mathcal{L}(F)$ e poi si calcolano i valori di verità delle altre sottoformule utilizzando le due tavole precedenti. Per esempio la tavola di verità di $\neg(\neg A \wedge \neg B)$ (ovvero di $A \vee B$) è

A	B	$\neg A$	$\neg B$	$(\neg A \wedge \neg B)$	$\neg(\neg A \wedge \neg B)$
0	0	1	1	1	0
0	1	1	0	0	1
1	0	0	1	0	1
1	1	0	0	0	1

Dunque la formula abbreviata da $A \vee B$ è vera esattamente quando almeno una tra A e B è vera (cioè ha lo stesso significato del “vel” latino, come desiderato). Allo stesso modo si possono ricavare le tavole di verità di tutti i connettivi diversi da \neg e \wedge (che riportiamo di seguito) e osservare che esse coincidono con il significato che volevamo attribuire a ciascuno di questi simboli (vedi Introduzione).

A	B	$A \vee B$
0	0	0
0	1	1
1	0	1
1	1	1

A	B	$A \rightarrow B$
0	0	1
0	1	1
1	0	0
1	1	1

A	B	$A \leftrightarrow B$
0	0	1
0	1	0
1	0	0
1	1	1

A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

A	B	$A \downarrow B$
0	0	1
0	1	0
1	0	0
1	1	0

Come abbiamo osservato all'inizio del Capitolo, abbiamo scelto di utilizzare \neg e \wedge come connettivi "primitivi" e di definire gli altri sulla base di questi: ma la scelta di utilizzare \neg e \wedge è (quasi) del tutto arbitraria. Volendo si sarebbero potuti scegliere altri connettivi (accompagnati dalla corrispondente tavola di verità che ne definisce la semantica) e definire i restanti sulla base di quelli prescelti. Più in generale un sottoinsieme \mathcal{B} dell'insieme $\mathcal{C} = \{\neg, \wedge, \vee, \rightarrow, \leftrightarrow, \oplus, \downarrow\}$ dei connettivi si dice *base* se per ogni connettivo non appartenente a tale insieme esiste una formula contenente solo i connettivi di \mathcal{B} tale che l'ultima colonna della sua tavola di verità sia uguale all'ultima colonna di quella corrispondente al connettivo prescelto⁴.

Ovviamente \mathcal{C} è una base. Inoltre abbiamo già visto che anche $\mathcal{B} = \{\neg, \wedge\}$ è una base poiché siamo stati in grado di definire gli altri connettivi a partire da quelli di \mathcal{B} , ma avremmo potuto scegliere qualunque altra base per descrivere la logica proposizionale senza perdere nulla del suo potere espressivo.

Esercizi.

1. Verificare che sono basi i seguenti insiemi: $\{\neg, \wedge\}$, $\{\neg, \vee\}$, $\{\neg, \rightarrow\}$ e $\{\downarrow\}$.

⁴Più precisamente, \mathcal{B} è una base se per ogni connettivo c anche a n argomenti, cA_1, \dots, A_n è logicamente equivalente (nel senso che definiremo più avanti) ad una formula contenente solo A_1, \dots, A_n , parentesi e connettivi di \mathcal{B} .

Sia ora $\mathcal{S} = \{A_1, \dots, A_n, \dots\}$ un insieme (finito o infinito) di formule atomiche (ovvero fissiamo il linguaggio $\mathcal{L} = \{\neg, \wedge, \vee, \rightarrow, \leftrightarrow, A_1, \dots, A_n, \dots\}$), e sia $\mathcal{F}(\mathcal{S})$ l'insieme delle formule che si possono costruire a partire da esse (ovvero l'insieme delle formule del linguaggio \mathcal{L}).

Un *assegnamento* su \mathcal{S} (o su \mathcal{L}) è definito come una funzione

$$\mathcal{A} : \mathcal{S} \rightarrow \{0, 1\}.$$

Tale funzione si estende in maniera naturale a tutte le formule di $\mathcal{F}(\mathcal{S})$. Infatti, se \mathcal{A} è definita su due formule F e G , possiamo estenderla a $(F \wedge G)$ ponendo $\mathcal{A}(F \wedge G) = 1$ se $\mathcal{A}(F) = \mathcal{A}(G) = 1$ e $\mathcal{A}(F \wedge G) = 0$ altrimenti. Inoltre possiamo definire $\mathcal{A}(\neg F) = 0$ se $\mathcal{A}(F) = 1$ e $\mathcal{A}(\neg F) = 1$ altrimenti.

Osservazione. Data una formula F , ogni assegnamento sulle sue sottoformule atomiche (ovvero su $\mathcal{L}(F)$) corrisponde ad una riga della tavola di verità: dunque calcolare la tavola di verità di F è equivalente a considerare tutti i possibili assegnamenti (sulle sue sottoformule atomiche).

Definizione 4. Dato un assegnamento \mathcal{A} su \mathcal{S} ed una formula $F \in \mathcal{F}(\mathcal{S})$, diremo che \mathcal{A} modella F (o, equivalentemente, che F è vera rispetto ad \mathcal{A} , o che \mathcal{A} soddisfa F) se $\mathcal{A}(F) = 1$. In questo caso scriveremo $\mathcal{A} \models F$.

F si dirà soddisfacibile se esiste un assegnamento \mathcal{A} tale che $\mathcal{A}(F) = 1$.

F si dirà valida (o che F è una tautologia, o che è sempre vera) se per ogni assegnamento \mathcal{A} si ha $\mathcal{A}(F) = 1$. In questo caso scriveremo $\models F$.

F si dirà insoddisfacibile (o che F è una contraddizione) se non è soddisfacibile, cioè se per ogni assegnamento \mathcal{A} si ha $\mathcal{A}(F) = 0$.

Osservazione. È evidente che F è una tautologia sse $\neg F$ è una contraddizione. Inoltre F è soddisfacibile sse F non è una contraddizione, e ogni tautologia è anche soddisfacibile.

Infine, F è valida (rispettivamente, soddisfacibile o contraddizione) se nella colonna terminale della sua tavola di verità compare *solo* l'1 (risp., *almeno* un 1, *nessun* 1).

Definizione 5. Si dice che G è conseguenza (logica) di F (e si scrive $F \models G$) se ogni assegnamento che modella F modella anche G . In simboli, $F \models G$ sse per ogni assegnamento \mathcal{A} , se $\mathcal{A} \models F$ allora $\mathcal{A} \models G$.

F e G si dicono (logicamente) equivalenti (e si scrive $F \equiv G$) sse sono una conseguenza logica dell'altra, cioè se $F \models G$ e $G \models F$.

Proposizione 1. $F \models G$ se e solo se $\models F \rightarrow G$.

$F \equiv G$ se e solo se $\models F \leftrightarrow G$.

Dimostrazione. Dimosteremo che $F \rightarrow G$ non è una tautologia se e solo se G non è una conseguenza logica di F . Per definizione di "tautologia",

$F \rightarrow G$ non è una tautologia se esiste un assegnamento \mathcal{A} tale che $\mathcal{A} \models \neg(F \rightarrow G)$. Per la semantica definita prima, ciò accade se e solo se sia $\mathcal{A} \models F$ che $\mathcal{A} \models \neg G$, ovvero se e solo se esiste un assegnamento che modella F ma non G . Ma per definizione di “conseguenza logica” questo equivale a dire che G non è conseguenza logica di F . \square

Per analogia, se \mathcal{F} è un insieme di formule e G è un'altra formula, diremo che G è conseguenza logica di \mathcal{F} (e scriveremo $\mathcal{F} \models G$) se per ogni assegnamento \mathcal{A} che modella *tutte* le formule di \mathcal{F} si ha $\mathcal{A}(G) = 1$.

È evidente che se \mathcal{F} è finito, indicando con $\bigwedge \mathcal{F}$ la congiunzione di tutte le formule che compaiono in \mathcal{F} , si ha

$$\mathcal{F} \models G \text{ sse } \bigwedge \mathcal{F} \models G \text{ sse } \bigwedge \mathcal{F} \rightarrow G.$$

Esercizi.

1. Costruire le tavole di verità delle seguenti proposizioni:

$$(A \rightarrow A) \rightarrow A$$

$$A \rightarrow (A \rightarrow A)$$

$$A \vee B \rightarrow A \wedge B$$

$$A \vee (B \wedge C) \rightarrow (A \wedge C) \vee D$$

$$A \rightarrow (B \rightarrow A)$$

2. Costruire le tavole di verità delle seguenti formule e dire se sono tautologie, contraddizioni o nessuna delle due:

$$(\neg A \rightarrow B) \vee ((A \wedge \neg C) \leftrightarrow B)$$

$$(A \rightarrow B) \wedge (A \rightarrow \neg B)$$

$$(A \rightarrow (B \vee C)) \vee (C \rightarrow \neg A)$$

$$((A \rightarrow B) \wedge C) \vee (A \wedge D)$$

3. Per ciascuna delle coppie di formule seguenti dire se sono equivalenti tra di loro:

$$(A \wedge B) \vee C \text{ e } (A \rightarrow \neg B) \rightarrow C$$

$$((A \rightarrow B) \rightarrow B) \rightarrow B \text{ e } A \rightarrow B$$

$$((A \rightarrow B) \rightarrow A) \rightarrow A \text{ e } (C \rightarrow D) \vee C$$

$$A \leftrightarrow ((\neg A \wedge B) \vee (A \wedge \neg B)) \text{ e } \neg B$$

$$A \vee (\neg B \wedge C) \text{ e } B \vee \neg C \rightarrow A$$

4. Stabilire se:

$$A \rightarrow B \wedge C \models (A \rightarrow B) \rightarrow C$$

5. Sia $\mathcal{F} = \{\neg A \rightarrow B, B \rightarrow C, \neg C \vee A\}$. Stabilire se:

$$\mathcal{F} \models A$$

6. Mostrare che le seguenti affermazioni sono equivalenti:

(a) $F \models G$

(b) $\models F \rightarrow G$

(c) $F \wedge \neg G$ è insoddisfacibile

(d) $F \equiv F \wedge G$.

7. Mostrare che le seguenti affermazioni sono equivalenti:

(a) $F \equiv G$

(b) $\models F \leftrightarrow G$

(c) $(F \wedge \neg G) \vee (\neg F \wedge G)$ è insoddisfacibile.

8. Stabilire se le seguenti formule sono valide:

$$(A \vee B) \wedge (B \rightarrow C) \rightarrow (\neg C \rightarrow A)$$

$$(A \rightarrow B) \wedge (A \rightarrow (B \rightarrow C)) \rightarrow (A \rightarrow C)$$

$$(A \rightarrow B) \rightarrow C \leftrightarrow A \wedge B \rightarrow C$$

$$A \rightarrow (B \rightarrow C) \leftrightarrow A \wedge B \rightarrow C$$

9. Verificare la validità delle seguenti leggi logiche notevoli:

Legge dell'identità	$A \rightarrow A$
Legge della doppia negazione	$A \leftrightarrow \neg\neg A$
Commutatività di \wedge	$A \wedge B \leftrightarrow B \wedge A$
Associatività di \wedge	$(A \wedge B) \wedge C \leftrightarrow A \wedge (B \wedge C)$
Commutatività di \vee	$A \vee B \leftrightarrow B \vee A$
Associatività di \vee	$(A \vee B) \vee C \leftrightarrow A \vee (B \vee C)$
Idempotenza di \wedge	$A \wedge A \leftrightarrow A$
Idempotenza di \vee	$A \vee A \leftrightarrow A$
Eliminazione di \wedge	$A \wedge B \rightarrow A$
Introduzione di \vee	$A \rightarrow A \vee B$
Distributività	$A \wedge (B \vee C) \leftrightarrow (A \wedge B) \vee (A \wedge C)$
Distributività	$A \vee (B \wedge C) \leftrightarrow (A \vee B) \wedge (A \vee C)$
Legge di assorbimento	$A \wedge (A \vee B) \leftrightarrow A$
Legge di assorbimento	$A \vee (A \wedge B) \leftrightarrow A$
Legge di De Morgan	$\neg(A \wedge B) \leftrightarrow (\neg A \vee \neg B)$
Legge di De Morgan	$\neg(A \vee B) \leftrightarrow (\neg A \wedge \neg B)$

Legge del terzo escluso	$A \vee \neg A$
Legge di non contraddizione	$\neg(A \wedge \neg A)$
Legge di contrapposizione	$A \rightarrow B \leftrightarrow \neg B \rightarrow \neg A$
Ex falso quodlibet	$A \wedge \neg A \rightarrow B$
Affermazione del conseguente	$A \rightarrow (B \rightarrow A)$
Negazione dell'antecedente	$\neg A \rightarrow (A \rightarrow B)$
Legge di riduzione all'assurdo	$(A \rightarrow B \wedge \neg B) \rightarrow \neg A$
Riduzione all'assurdo debole	$(A \rightarrow \neg A) \rightarrow \neg A$
Consequentia mirabilis	$(\neg A \rightarrow A) \rightarrow A$
Legge di Pierce	$((A \rightarrow B) \rightarrow A) \rightarrow A$
Legge di Dummett	$(A \rightarrow B) \vee (B \rightarrow A)$
Modus ponens	$A \rightarrow ((A \rightarrow B) \rightarrow B)$
Modus ponens	$A \wedge (A \rightarrow B) \rightarrow B$
Scambio antecedenti	$A \rightarrow (B \rightarrow C) \leftrightarrow B \rightarrow (A \rightarrow C)$
Distinzione di casi	$(A \rightarrow C) \wedge (B \rightarrow C) \leftrightarrow A \vee B \rightarrow C$
Distinzione di casi	$(A \rightarrow B) \wedge (\neg A \rightarrow B) \rightarrow B$
Distributività di \rightarrow	$(A \rightarrow (B \rightarrow C)) \rightarrow ((A \rightarrow B) \rightarrow (A \rightarrow C))$
Transitività di \rightarrow	$(A \rightarrow B) \wedge (B \rightarrow C) \rightarrow (A \rightarrow C)$
Import/export premesse	$A \rightarrow (B \rightarrow C) \leftrightarrow A \wedge B \rightarrow C$
Sillogismo disgiuntivo	$A \wedge (\neg A \vee B) \rightarrow B$

Osservazione. Il metodo delle tavole semantiche corrisponde ad un algoritmo che permette di determinare quando una formula sia valida (equivalentemente, quando sia una contraddizione, quando sia soddisfacibile, quando due formule siano logicamente equivalenti o una conseguenza logica dell'altra), ma non è efficiente poiché lavora in un tempo non polinomiale. Infatti, se nella formula compaiono n formule atomiche, è necessario calcolare 2^n righe della tavola di verità per avere una risposta.

1.3 Traduzione dal linguaggio naturale al linguaggio formale

Come abbiamo detto nell'introduzione, i linguaggi logici sono linguaggi artificiali che permettono di "tradurre" (cioè *formalizzare*) frasi del linguaggio naturale in formule. In questa Sezione vedremo come tradurre frasi del linguaggio naturale in formule proposizionali. Innanzitutto, a meno che questo non venga già dato nel testo, è necessario associare una diversa lettera proposizionale ad ogni proposizione semplice (nel senso dell'analisi logica del testo) che compare nella frase. Dopodiché si passa a sostituire tutte le varie particelle grammaticali che collegano (o negano) le proposizioni con i

corrispettivi connettivi della logica proposizionale (facendo attenzione al significato!). Alle volte è possibile formalizzare un intero ragionamento (cioè un insieme di frasi costituito da alcune premesse e una conclusione) semplicemente formalizzando le varie frasi, congiungendo (con il simbolo \wedge) le premesse e poi costruendo un'implicazione ponendo la congiunzione delle premesse a sinistra del simbolo \rightarrow e la (formalizzazione della) conclusione a destra. In questo caso ci si può anche chiedere se il ragionamento è corretto no, il che equivale a chiedere se la sua formalizzazione (che è una formula della logica proposizionale) sia una tautologia o no: per rispondere a questa domanda, come abbiamo visto, sarà sufficiente calcolarne la tavola di verità.

Esercizi.

1. Consideriamo un linguaggio proposizionale in cui P significa “Pietro verrà eletto leader del partito”, R significa “Roberta si dimetterà”, M significa “Mario si dimetterà” e V significa “vinceremo le elezioni”. Tradurre le seguenti frasi nel linguaggio formale:

Vinceremo le elezioni, se Pietro sarà eletto leader del partito.

Vinceremo le elezioni solo se Pietro sarà eletto leader del partito (equivalente a “se Pietro non verrà eletto leader del partito perderemo certamente le elezioni”).

Se Pietro non sarà eletto leader del partito, allora Roberta o Mario si dimetteranno e non vinceremo le elezioni.

2. Introducendo opportuni linguaggi proposizionali, tradurre le frasi seguenti nel linguaggio formale:

- (a) Se l'algoritmo termina abbiamo un risultato, e se abbiamo un risultato lo stampiamo.

Se l'algoritmo termina abbiamo un risultato e lo stampiamo.

Non è possibile che l'algoritmo non termini.

Non è possibile che l'algoritmo termini ma non dia un risultato.

- (b) Paola va al cinema solo se Roberto ci va.

Se Roberto va al cinema, anche Paola ci va.

Al massimo uno tra Roberto e Paola va al cinema.

- (c) Se Giorgio è felice, Anna è felice.

Se Giorgio è infelice, Anna è infelice.

- (d) Data una funzione reale f , la continuità in un punto x è condizione necessaria per la derivabilità di f in x .

La continuità di f in x non è una condizione sufficiente affinché f sia derivabile in x .

3. Si consideri il problema del *merging* di due liste $List1$ e $List2$ in una terza lista $List3$ (ad esempio nomi, in ordine alfabetico).

Una prima formulazione dell'algoritmo è la seguente:

Nello scorrere le due liste, se $List1$ non è esaurita e, inoltre, o $List2$ è esaurita oppure l'elemento in considerazione di $List1$ precede il primo non ancora inserito di $List2$, allora l'elemento di $List1$ è inserito in $List3$.

Un'altra formulazione potrebbe essere la seguente:

il prossimo elemento di $List3$ è preso da $List1$ quando $List1$ non è esaurita e $List2$ sì, oppure quando $List1$ non è esaurita e l'elemento in considerazione di $List1$ precede il primo non ancora inserito di $List2$.

Introducendo un opportuno linguaggio proposizionale, scrivere le proposizioni corrispondenti alle due versioni delle condizioni (che portano entrambe a mettere in $List3$ l'elemento in esame di $List1$), e discutere se siano o no equivalenti e in base a quali leggi.

4. Si distribuiscono carte da gioco, e si sa che un giocatore ha in mano un Asso o un Re. Si considerino le seguenti due proposizioni:

F : se c'è in mano un Asso, c'è anche un 2

G : se c'è in mano un Re, c'è anche un 2.

Che cosa si può dedurre se esattamente una tra le proposizioni F e G è vera?

Che cosa si può dedurre se entrambi le proposizioni F e G sono vere?

5. Aladino trova in una grotta due bauli A e B e sa che ciascuno di essi o contiene un tesoro, oppure contiene una trappola mortale che lo ucciderebbe all'istante. Sul baule A è scritto:

Almeno uno di questi bauli contiene un tesoro.

Sul baule B è scritto:

Nel baule A c'è una trappola mortale che scatta quando il baule viene aperto e uccide all'istante.

Ad Aladino viene detto che o entrambe le scritte sono vere, o entrambe false. Quale baule deve aprire Aladino per trovare il tesoro?

6. Se io ho ragione, tu hai torto; se tu hai ragione io ho torto. Quindi uno di noi ha ragione.

Corretto o no? Perché?

1.4 Dimostrazioni formali

Siano F , G ed H formule proposizionali e siano \mathcal{F} ed \mathcal{F}' due insiemi di formule (proposizionali). Consideriamo il *sistema di derivazione formale* seguente:

Premesse	Conclusione	Nome
$G \in \mathcal{F}$	$\mathcal{F} \vdash G$	Assunzione
$\mathcal{F} \vdash G$ e $\mathcal{F} \subseteq \mathcal{F}'$	$\mathcal{F}' \vdash G$	Monotonia
$\mathcal{F} \vdash G$	$\mathcal{F} \vdash \neg\neg G$	Doppia negazione
$\mathcal{F} \vdash F$ e $\mathcal{F} \vdash G$	$\mathcal{F} \vdash F \wedge G$	\wedge -introduzione
$\mathcal{F} \vdash F \wedge G$	$\mathcal{F} \vdash F$	\wedge -eliminazione
$\mathcal{F} \vdash F \wedge G$	$\mathcal{F} \vdash G \wedge F$	\wedge -simmetria
$\mathcal{F} \vdash F$	$\mathcal{F} \vdash F \vee G$	\vee -introduzione
$\mathcal{F} \vdash F \vee G$ e $\mathcal{F} \cup \{F\} \vdash H$ e $\mathcal{F} \cup \{G\} \vdash H$	$\mathcal{F} \vdash H$	\vee -eliminazione
$\mathcal{F} \vdash F \vee G$	$\mathcal{F} \vdash G \vee F$	\vee -simmetria
$\mathcal{F} \cup \{F\} \vdash G$	$\mathcal{F} \vdash F \rightarrow G$	\rightarrow -introduzione ⁵
$\mathcal{F} \vdash F \rightarrow G$ e $\mathcal{F} \vdash F$	$\mathcal{F} \vdash G$	\rightarrow -eliminazione ⁶

Inoltre, avendo scelto \neg e \wedge come connettivi primitivi, aggiungiamo le seguenti regole:

$\mathcal{F} \vdash F \vee G$	se e solo se	$\mathcal{F} \vdash \neg(\neg F \wedge \neg G)$	\vee -definizione
$\mathcal{F} \vdash F \rightarrow G$	se e solo se	$\mathcal{F} \vdash \neg F \vee G$	\rightarrow -definizione
$\mathcal{F} \vdash F \leftrightarrow G$	se e solo se	sia $\mathcal{F} \vdash F \rightarrow G$, sia $\mathcal{F} \vdash G \rightarrow F$	\leftrightarrow -definizione

Definizione 6. Una dimostrazione (o derivazione) formale è una sequenza finita di affermazioni del tipo $\mathcal{F} \vdash F$, ognuna delle quali segue dalle precedenti per una delle regole di derivazione del sistema formale.

La lunghezza di tale sequenza verrà detta altezza della derivazione.

Diremo che G è derivabile da \mathcal{F} se esiste una dimostrazione formale che termina con $\mathcal{F} \vdash G$.

G si dirà derivabile se è derivabile da un insieme vuoto di formule, cioè se $\emptyset \vdash G$. In questo caso scriveremo più semplicemente $\vdash G$.

Quando si vuole dimostrare qualcosa, normalmente si hanno delle *premesse* (le ipotesi) e una formula da dimostrare (*conclusione* o tesi). Allora la dimostrazione formale potrà utilizzare liberamente le premesse e derivare la conclusione con le regole di derivazione formale. Ogni volta che si dimostra qualcosa di questo tipo, le premesse e la conclusione possono essere inserite

nella lista delle regole di derivazione e utilizzate per abbreviare successive dimostrazioni. Particolarmente importanti (perché vengono usate spesso nelle dimostrazioni matematiche) sono:

Premesse	Conclusione	Nome
$\mathcal{F} \vdash \neg F \vee G$ e $\mathcal{F} \vdash F$	$\mathcal{F} \vdash G$	\vee -modus ponens
Nessuna	$\mathcal{F} \vdash \neg G \vee G$	Terzo escluso
$\mathcal{F} \vdash F \wedge \neg F$	$\mathcal{F} \vdash G$	Contraddizione
$\mathcal{F} \cup \{F\} \vdash G$	$\mathcal{F} \cup \{\neg G\} \vdash \neg F$	Contrapposizione
$\mathcal{F} \cup \{F\} \vdash G$ e $\mathcal{F} \cup \{\neg F\} \vdash G$	$\mathcal{F} \vdash G$	Dimostrazione per casi
$\mathcal{F} \cup \{F\} \vdash G$ e $\mathcal{F} \cup \{F\} \vdash \neg G$	$\mathcal{F} \vdash \neg F$	Dimostrazione per assurdo (o per contraddizione)

Chiaramente tali regole possono essere derivate da quelle base, per esempio mediante le seguenti dimostrazioni formali.

1. 1. $\mathcal{F} \vdash \neg F \vee G$ Premessa
2. $\mathcal{F} \vdash F$ Premessa
3. $\mathcal{F} \vdash F \rightarrow G$ \rightarrow -Def su 1
4. $\mathcal{F} \vdash G$ \rightarrow -E su 2 e 3

2. 1. $\mathcal{F} \cup \{G\} \vdash G$ Ass
2. $\mathcal{F} \vdash G \rightarrow G$ \rightarrow -I su 1
3. $\mathcal{F} \vdash \neg G \vee G$ \rightarrow -Def su 2

3. 1. $\mathcal{F} \vdash F \wedge \neg F$ Premessa⁷
2. $\mathcal{F} \vdash \neg F \wedge F$ \wedge -Simm su 1
3. $\mathcal{F} \vdash \neg F$ \wedge -E su 2
4. $\mathcal{F} \vdash \neg F \vee G$ \vee -I su 3
5. $\mathcal{F} \vdash F \rightarrow G$ \rightarrow -Def su 4
6. $\mathcal{F} \vdash F$ \wedge -E su 1
7. $\mathcal{F} \vdash G$ \rightarrow -E su 5 e 6

4. 1. $\mathcal{F} \cup \{F\} \vdash G$ Premessa
2. $\mathcal{F} \cup \{F\} \vdash \neg \neg G$ DN su 1
3. $\mathcal{F} \vdash F \rightarrow \neg \neg G$ \rightarrow -I su 2
4. $\mathcal{F} \vdash \neg F \vee \neg \neg G$ \rightarrow -Def su 3
5. $\mathcal{F} \vdash \neg \neg G \vee \neg F$ \vee -Simm su 4

⁷Questa regola viene anche chiamata “ex falso quodlibet”.

6. $\mathcal{F} \vdash \neg G \rightarrow \neg F$ \rightarrow -Def su 5
 7. $\mathcal{F} \cup \{\neg G\} \vdash \neg G$ Ass
 8. $\mathcal{F} \cup \{\neg G\} \vdash \neg G \rightarrow \neg F$ Mon su 7
 9. $\mathcal{F} \cup \{\neg G\} \vdash \neg F$ \rightarrow -E su 7 e 8
5. 1. $\mathcal{F} \cup \{F\} \vdash G$ Premessa
 2. $\mathcal{F} \cup \{\neg F\} \vdash G$ Premessa
 3. $\mathcal{F} \vdash F \vee \neg F$ TE
 4. $\mathcal{F} \vdash G$ *vee*-E su 3, 1 e 2
6. 1. $\mathcal{F} \cup \{F\} \vdash G$ Premessa
 2. $\mathcal{F} \cup \{F\} \vdash \neg G$ Premessa
 3. $\mathcal{F} \cup \{F\} \vdash G \wedge \neg G$ \wedge -I su 1 e 2
 4. $\mathcal{F} \cup \{F\} \vdash \neg F$ Contraddizione su 3
 5. $\mathcal{F} \cup \{\neg F\} \vdash \neg F$ Ass
 6. $\mathcal{F} \vdash \neg F$ Casi su 4 e 5

Definizione 7. Due formule F e G si dicono dimostrabilmente equivalenti (in simboli $F \dashv\vdash G$) se sia $\{F\} \vdash G$ che $\{G\} \vdash F$.

Teorema 2 (Correttezza (o solidità) del sistema formale). Se $\mathcal{F} \vdash G$ allora $\mathcal{F} \vDash G$.

Dimostrazione. Se $\mathcal{F} \vdash G$ allora per definizione esiste una dimostrazione formale che termina con $\mathcal{F} \vdash G$. Ogni riga della dimostrazione contiene un'affermazione del tipo $\mathcal{X} \vdash Y$ giustificata da una delle regole del sistema formale (prime due tabelle). Dimostreremo che per ogni linea della dimostrazione, se $\mathcal{X} \vdash Y$ allora $\mathcal{X} \vDash Y$. Questo risultato si ottiene verificando tale affermazione per ciascuna delle regole di derivazione. Considereremo qui alcuni casi, lasciando la verifica dei restanti come esercizio per il lettore.

Consideriamo la regola di assunzione, e supponiamo $G \in \mathcal{F}$. Allora per ogni assegnamento \mathcal{A} , se $\mathcal{A} \vDash \mathcal{F}$ allora in particolare $\mathcal{A} \vDash G$, e dunque $\mathcal{F} \vDash G$.

Passiamo ora alla regola di eliminazione del \wedge . Ciò che dobbiamo dimostrare in questo caso è che se $\mathcal{F} \vDash F \wedge G$ allora $\mathcal{F} \vdash F$. Questo segue da $F \wedge G \vDash F$, poiché per ogni assegnamento \mathcal{A} tale che $\mathcal{A} \vDash \mathcal{F}$ si avrebbe che $\mathcal{A} \vDash F \wedge G$ e quindi $\mathcal{A} \vDash F$ per le nostre ipotesi. Per dimostrare che $F \wedge G \vDash F$ possiamo dimostrare che $F \wedge G \rightarrow F$ è una tautologia mediante la seguente tavola di verità:

F	G	$F \wedge G$	$F \wedge G \rightarrow F$
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	1

Infine, consideriamo il caso dell'introduzione del \rightarrow . Per verificare questa regola bisogna dimostrare che se $\mathcal{F} \cup \{F\} \vDash G$ allora $\mathcal{F} \vDash F \rightarrow G$. Sia allora \mathcal{A} un assegnamento tale che $\mathcal{A} \vDash \mathcal{F}$. Possono presentarsi due casi: se $\mathcal{A} \vDash F$ allora $\mathcal{A} \vDash G$ (per le nostre ipotesi) e quindi $\mathcal{A} \vDash F \rightarrow G$ (poiché antecedente e conseguente dell'implicazione sono vere). Se invece $\mathcal{A} \not\vDash F$ allora $\mathcal{A} \vDash F \rightarrow G$ indipendentemente dal valore di $\mathcal{A}(G)$ (poiché l'antecedente dell'implicazione è falso). Dunque, in ogni caso $\mathcal{A} \vDash F \rightarrow G$ e quindi $\mathcal{F} \vDash F \rightarrow G$.

In sostanza, quindi, bisogna verificare che tutte le regole delle prime due tabelle sono valide quando si sostituisca il simbolo \vdash con \vDash . \square

Corollario 3. *Se G è derivabile (da $\mathcal{F} = \emptyset$) allora G è una tautologia.*

Se $\neg G$ è derivabile allora G è una contraddizione.

Se F e G sono dimostrabilmente equivalenti allora sono (logicamente) equivalenti.

Esercizi.

1. Dimostrare che G è derivabile da F (cioè $\{F\} \vdash G$) se e solo se per ogni insieme di formule \mathcal{F} , se $\mathcal{F} \vdash F$ allora $\mathcal{F} \vdash G$.

Dimostrazione. Assumiamo $\{F\} \vdash G$ e $\mathcal{F} \vdash F$ e deriviamo formalmente $\mathcal{F} \vdash G$.

1. $\mathcal{F} \vdash F$ Premessa
2. $\{F\} \vdash G$ Premessa
3. $\mathcal{F} \cup \{F\} \vdash G$ Mon su 2
4. $\mathcal{F} \vdash F \rightarrow G$ \rightarrow -I su 3
5. $\mathcal{F} \vdash G$ \rightarrow -E su 4

Viceversa, se $\mathcal{F} \vdash F$ implica $\mathcal{F} \vdash G$ per ogni insieme di formule \mathcal{F} , allora possiamo considerare $\mathcal{F} = \{F\}$ e concludere $\{F\} \vdash G$ poiché $\{F\} \vdash F$ per la regola di assunzione. \square

2. Dimostrare che $\mathcal{F} \vdash F \wedge G$ se e solo se $\mathcal{F} \vdash \neg(\neg F \vee \neg G)$.
3. Dare dimostrazioni formali delle seguenti regole:
 - (a) Premessa: $\mathcal{F} \vdash F \rightarrow G$
Conclusione: $\mathcal{F} \cup \{F\} \vdash G$
 - (b) Premessa: $\mathcal{F} \vdash F$
Conclusione: $\mathcal{F} \cup \{\neg F\} \vdash G$

(c) **Eliminazione della Doppia Negazione**Premessa: $\mathcal{F} \vdash \neg\neg F$ Conclusione: $\mathcal{F} \vdash F$ *Soluzione:*

1. $\mathcal{F} \vdash \neg\neg F$ Premessa
2. $\mathcal{F} \cup \{\neg F\} \vdash \neg F$ Ass
3. $\mathcal{F} \cup \{\neg F\} \vdash \neg\neg F$ Mon su 1
4. $\mathcal{F} \cup \{\neg F\} \vdash \neg F \wedge \neg\neg F$ \wedge -I su 2 e 3
5. $\mathcal{F} \cup \{\neg F\} \vdash F$ Contraddizione su 4
6. $\mathcal{F} \cup \{F\} \vdash F$ Ass
7. $\mathcal{F} \vdash F$ Casi su 5 e 6

(d) **\vee -modus ponens₂**Premesse: $\mathcal{F} \vdash F \vee G$ e $\mathcal{F} \vdash \neg F$ Conclusione: $\mathcal{F} \vdash G$ *Soluzione:*

1. $\mathcal{F} \vdash F \vee G$ Premessa
2. $\mathcal{F} \vdash \neg F$ Premessa
3. $\mathcal{F} \cup \{F\} \vdash \neg F$ Mon su 2
4. $\mathcal{F} \cup \{F\} \vdash F$ Ass
5. $\mathcal{F} \cup \{F\} \vdash F \wedge \neg F$ \wedge -I su 3 e 4
6. $\mathcal{F} \cup \{F\} \vdash G$ Contraddizione su 5
7. $\mathcal{F} \cup \{G\} \vdash G$ Ass
8. $\mathcal{F} \vdash G$ \vee -E su 1, 6 e 7

(e) **Definizione di \wedge .** $\mathcal{F} \vdash F \wedge G$ se e solo se $\mathcal{F} \vdash \neg(\neg F \vee \neg G)$.*Soluzione:*

Bisogna fare due dimostrazioni formali:

1. $\mathcal{F} \vdash F \wedge G$ Premessa
2. $\mathcal{F} \cup \{\neg F \vee \neg G\} \vdash F \wedge G$ Mon su 1
3. $\mathcal{F} \cup \{\neg F \vee \neg G\} \vdash \neg F \vee \neg G$ Ass
4. $\mathcal{F} \cup \{\neg F \vee \neg G\} \vdash F$ \wedge -E su 2
5. $\mathcal{F} \cup \{\neg F \vee \neg G\} \vdash \neg G$ \vee -MP su 4 e 3
6. $\mathcal{F} \cup \{\neg F \vee \neg G\} \vdash G \wedge F$ \wedge -Simm su 2
7. $\mathcal{F} \cup \{\neg F \vee \neg G\} \vdash G$ \wedge -E su 6
8. $\mathcal{F} \vdash \neg(\neg F \vee \neg G)$ Assurdo su 5 e 7

1. $\mathcal{F} \vdash \neg(\neg F \vee \neg G)$ Premessa
2. $\mathcal{F} \cup \{\neg F\} \vdash \neg(\neg F \vee \neg G)$ Mon su 1

3. $\mathcal{F} \cup \{\neg F\} \vdash \neg F$ Ass
4. $\mathcal{F} \cup \{\neg F\} \vdash \neg F \vee \neg G$ \vee -I su 3
5. $\mathcal{F} \vdash \neg\neg F$ Assurdo su 2 e 4
6. $\mathcal{F} \vdash F$ DN-E su 5
7. $\mathcal{F} \cup \{\neg G\} \vdash \neg(\neg F \vee \neg G)$ Mon su 1
8. $\mathcal{F} \cup \{\neg G\} \vdash \neg G$ Ass
9. $\mathcal{F} \cup \{\neg G\} \vdash \neg G \vee \neg F$ \vee -I su 8
10. $\mathcal{F} \cup \{\neg G\} \vdash \neg F \vee \neg G$ \vee -Simm su 9
11. $\mathcal{F} \vdash \neg\neg G$ Assurdo su 7 e 10
12. $\mathcal{F} \vdash G$ DN-E su 11
13. $\mathcal{F} \vdash F \wedge G$ \wedge -I su 6 e 12

(f) **Regole di De Morgan.** Le seguenti coppie di formule sono dimostrabilmente equivalenti:

$$\neg(F \wedge G) \text{ e } \neg F \vee \neg G$$

$$\neg(F \vee G) \text{ e } \neg F \wedge \neg G.$$

Soluzione:

Bisogna effettuare quattro dimostrazioni formali:

1. $\{\neg(F \wedge G)\} \vdash \neg(F \wedge G)$ Assunzione
 2. $\{\neg(F \wedge G), F, G\} \vdash \neg(F \wedge G)$ Mon su 1
 3. $\{\neg(F \wedge G), F, G\} \vdash F$ Ass
 4. $\{\neg(F \wedge G), F, G\} \vdash G$ Ass
 5. $\{\neg(F \wedge G), F, G\} \vdash F \wedge G$ \wedge -I su 3 e 4
 6. $\{\neg(F \wedge G), F\} \vdash \neg G$ Assurdo su 2 e 5
 7. $\{\neg(F \wedge G), F\} \vdash \neg G \vee \neg F$ \vee -I su 6
 8. $\{\neg(F \wedge G), F\} \vdash \neg F \vee \neg G$ \vee -Simm su 7
 9. $\{\neg(F \wedge G), \neg F\} \vdash \neg F$ Ass
 10. $\{\neg(F \wedge G), \neg F\} \vdash \neg F \vee \neg G$ \vee -I su 9
 11. $\{\neg(F \wedge G)\} \vdash \neg F \vee \neg G$ Casi su 8 e 10
-
1. $\{\neg F \vee \neg G, F \wedge G\} \vdash \neg F \vee \neg G$ Ass
 2. $\{\neg F \vee \neg G, F \wedge G\} \vdash F \wedge G$ Ass
 3. $\{\neg F \vee \neg G, F \wedge G\} \vdash \neg(\neg F \vee \neg G)$ \wedge -Def su 2
 4. $\{\neg F \vee \neg G\} \vdash \neg(F \wedge G)$ Assurdo su 1 e 3
-
1. $\{\neg(F \vee G)\} \vdash \neg(F \vee G)$ Ass
 2. $\{\neg(F \vee G), F\} \vdash \neg(F \vee G)$ Mon su 1
 3. $\{\neg(F \vee G), F\} \vdash F$ Ass
 4. $\{\neg(F \vee G), F\} \vdash F \vee G$ \vee -I su 3
 5. $\{\neg(F \vee G)\} \vdash \neg F$ Assurdo su 2 e 4

6. $\{\neg(F \vee G), G\} \vdash \neg(F \vee G)$ Mon su 1
7. $\{\neg(F \vee G), G\} \vdash G$ Ass
8. $\{\neg(F \vee G), G\} \vdash G \vee F$ \vee -I su 7
9. $\{\neg(F \vee G), G\} \vdash F \vee G$ \vee -Simm su 8
10. $\{\neg(F \vee G)\} \vdash \neg G$ Assurdo su 6 e 9
11. $\{\neg(F \vee G)\} \vdash \neg F \wedge \neg G$ \wedge -I su 5 e 10

1. $\{\neg F \wedge \neg G, F \vee G\} \vdash \neg F \wedge \neg G$ Ass
2. $\{\neg F \wedge \neg G, F \vee G\} \vdash F \vee G$ Ass
3. $\{\neg F \wedge \neg G, F \vee G\} \vdash \neg(\neg F \wedge \neg G)$ \vee -Def su 2
4. $\{\neg F \wedge \neg G\} \vdash \neg(F \vee G)$ Assurdo su 1 e 3

(g) **Distributività di \wedge e \vee .** Le seguenti coppie di formule sono dimostrabilmente equivalenti:

$F \wedge (G \vee H)$ e $(F \wedge G) \vee (F \wedge H)$

$F \vee (G \wedge H)$ e $(F \vee G) \wedge (F \vee H)$

Soluzione:

Dobbiamo fare quattro dimostrazioni formali:

1. $\mathcal{F} \vdash F \wedge (G \vee H)$ Premessa
2. $\mathcal{F} \vdash F$ \wedge -E su 1
3. $\mathcal{F} \vdash (G \vee H) \wedge F$ \wedge -Simm su 1
4. $\mathcal{F} \vdash G \vee H$ \wedge -E su 3
5. $\mathcal{F} \cup \{G\} \vdash F$ Mon su 2
6. $\mathcal{F} \cup \{G\} \vdash G$ Ass
7. $\mathcal{F} \cup \{G\} \vdash F \wedge G$ \wedge -I su 5 e 6
8. $\mathcal{F} \cup \{G\} \vdash (F \wedge G) \vee (F \wedge H)$ \vee -I su 7
9. $\mathcal{F} \cup \{H\} \vdash F$ Mon su 2
10. $\mathcal{F} \cup \{H\} \vdash H$ Ass
11. $\mathcal{F} \cup \{H\} \vdash F \wedge H$ \wedge -I su 9 e 10
12. $\mathcal{F} \cup \{H\} \vdash (F \wedge H) \vee (F \wedge G)$ \vee -I su 11
13. $\mathcal{F} \cup \{H\} \vdash (F \wedge G) \vee (F \wedge H)$ \vee -Simm su 12
14. $\mathcal{F} \vdash (F \wedge G) \vee (F \wedge H)$ \vee -E su 4, 8 e 13

1. $\mathcal{F} \vdash (F \wedge G) \vee (F \wedge H)$ Premessa
2. $\mathcal{F} \cup \{F \wedge G\} \vdash F \wedge G$ Ass
3. $\mathcal{F} \cup \{F \wedge G\} \vdash F$ \wedge -E su 2
4. $\mathcal{F} \cup \{F \wedge G\} \vdash G \wedge F$ \wedge -Simm su 2
5. $\mathcal{F} \cup \{F \wedge G\} \vdash G$ \wedge -E su 4
6. $\mathcal{F} \cup \{F \wedge G\} \vdash G \vee H$ \vee -I su 5
7. $\mathcal{F} \cup \{F \wedge G\} \vdash F \wedge (G \vee H)$ \wedge -I su 3 e 6

8. $\mathcal{F} \cup \{F \wedge H\} \vdash F \wedge H$ Ass
 9. $\mathcal{F} \cup \{F \wedge H\} \vdash F$ \wedge -E su 8
 10. $\mathcal{F} \cup \{F \wedge H\} \vdash H \wedge G$ \wedge -Simm su 8
 11. $\mathcal{F} \cup \{F \wedge H\} \vdash H$ \wedge -E su 10
 12. $\mathcal{F} \cup \{F \wedge H\} \vdash H \vee G$ \vee -I su 11
 13. $\mathcal{F} \cup \{F \wedge H\} \vdash G \vee H$ \vee -Simm su 12
 14. $\mathcal{F} \cup \{F \wedge H\} \vdash F \wedge (G \vee H)$ \wedge -I su 9 e 13
 15. $\mathcal{F} \vdash F \wedge (G \vee H)$ \vee -E su 1, 7 e 14
-
1. $\mathcal{F} \vdash F \vee (G \wedge H)$ Premessa
 2. $\mathcal{F} \cup \{F\} \vdash F$ Ass
 3. $\mathcal{F} \cup \{F\} \vdash F \vee G$ \vee -I su 2
 4. $\mathcal{F} \cup \{F\} \vdash F \vee H$ \vee -I su 2
 5. $\mathcal{F} \cup \{F\} \vdash (F \vee G) \wedge (F \vee H)$ \wedge -I su 3 e 4
 6. $\mathcal{F} \cup \{G \wedge H\} \vdash G \wedge H$ Ass
 7. $\mathcal{F} \cup \{G \wedge H\} \vdash G$ \wedge -E su 6
 8. $\mathcal{F} \cup \{G \wedge H\} \vdash G \vee F$ \vee -I su 7
 9. $\mathcal{F} \cup \{G \wedge H\} \vdash F \vee G$ \vee -Simm su 8
 10. $\mathcal{F} \cup \{G \wedge H\} \vdash H \wedge G$ \wedge -Simm su 6
 11. $\mathcal{F} \cup \{G \wedge H\} \vdash H$ \wedge -E su 10
 12. $\mathcal{F} \cup \{G \wedge H\} \vdash H \vee F$ \vee -I su 11
 13. $\mathcal{F} \cup \{G \wedge H\} \vdash F \vee H$ \vee -Simm su 12
 14. $\mathcal{F} \cup \{G \wedge H\} \vdash (F \vee G) \wedge (F \vee H)$ \wedge -I su 9 e 13
 15. $\mathcal{F} \vdash (F \vee G) \wedge (F \vee H)$ \vee -E su 1, 5 e 14
-
1. $\mathcal{F} \vdash (F \vee G) \wedge (F \vee H)$ Premessa
 2. $\mathcal{F} \cup \{\neg F\} \vdash (F \vee G) \wedge (F \vee H)$ Mon su 1
 3. $\mathcal{F} \cup \{\neg F\} \vdash F \vee G$ \wedge -E su 2
 4. $\mathcal{F} \cup \{\neg F\} \vdash \neg F$ Ass
 5. $\mathcal{F} \cup \{\neg F\} \vdash G$ \vee -MP₂ su 3 e 4
 6. $\mathcal{F} \cup \{\neg F\} \vdash (F \vee H) \wedge (F \vee G)$ \wedge -Simm su 2
 7. $\mathcal{F} \cup \{\neg F\} \vdash F \vee H$ \wedge -E su 6
 8. $\mathcal{F} \cup \{\neg F\} \vdash H$ \vee -MP₂ su 4 e 7
 9. $\mathcal{F} \cup \{\neg F\} \vdash G \wedge H$ \wedge -I su 5 e 8
 10. $\mathcal{F} \cup \{\neg F\} \vdash (G \wedge H) \vee F$ \vee -I su 9
 11. $\mathcal{F} \cup \{\neg F\} \vdash F \vee (G \wedge H)$ \vee -Simm su 10
 12. $\mathcal{F} \cup \{F\} \vdash F$ Ass
 13. $\mathcal{F} \cup \{F\} \vdash F \vee (G \wedge H)$ \vee -I su 12
 14. $\mathcal{F} \vdash F \vee (G \wedge H)$ Casi su 11 e 13

(h) **Consequentia mirabilis.**Premessa: $\mathcal{F} \vdash \neg F \rightarrow F$ Conclusione: $\mathcal{F} \vdash F$ *Soluzione:*

1. $\mathcal{F} \vdash \neg F \rightarrow F$ Premessa
2. $\mathcal{F} \cup \{\neg F\} \vdash \neg F \rightarrow F$ Mon su 1
3. $\mathcal{F} \cup \{\neg F\} \vdash \neg F$ Ass
4. $\mathcal{F} \cup \{\neg F\} \vdash F$ \rightarrow -E su 2 e 3
5. $\mathcal{F} \cup \{F\} \vdash F$ Ass
6. $\mathcal{F} \vdash F$ Casi su 4 e 5

(i) **Regola di taglio.** $\{F \rightarrow G, G \rightarrow H\} \vdash F \rightarrow H$ *Soluzione:*

1. $\{F \rightarrow G, G \rightarrow H, F\} \vdash F$ Ass
2. $\{F \rightarrow G, G \rightarrow H, F\} \vdash F \rightarrow G$ Ass
3. $\{F \rightarrow G, G \rightarrow H, F\} \vdash G$ \rightarrow -E su 1 e 2
4. $\{F \rightarrow G, G \rightarrow H, F\} \vdash G \rightarrow H$ Ass
5. $\{F \rightarrow G, G \rightarrow H, F\} \vdash H$ \rightarrow -E su 3 e 4
6. $\{F \rightarrow G, G \rightarrow H\} \vdash F \rightarrow H$ \rightarrow -I su 5

(j) **\leftrightarrow -simmetria.**Premessa: $\mathcal{F} \vdash F \leftrightarrow G$ Conclusione: $\mathcal{F} \vdash G \leftrightarrow F$ *Soluzione:*

1. $\mathcal{F} \vdash F \leftrightarrow G$ Premessa
2. $\mathcal{F} \vdash F \rightarrow G$ \leftrightarrow -Def su 1
3. $\mathcal{F} \vdash G \rightarrow F$ \leftrightarrow -Def su 1
4. $\mathcal{F} \vdash G \leftrightarrow F$ \leftrightarrow -Def su 3 e 2

(k) **\rightarrow -contrapposizione.**Premessa: $\mathcal{F} \vdash F \rightarrow G$ Conclusione: $\mathcal{F} \vdash \neg G \rightarrow \neg F$ *Soluzione:*

1. $\mathcal{F} \vdash F \rightarrow G$ Premessa
2. $\mathcal{F} \cup \{\neg G\} \vdash F \rightarrow G$ Mon su 1
3. $\mathcal{F} \cup \{\neg G\} \vdash \neg G$ Ass
4. $\mathcal{F} \cup \{\neg G\} \vdash \neg F \vee G$ \rightarrow -Def su 2
5. $\mathcal{F} \cup \{\neg G\} \vdash G \vee \neg F$ \vee -Simm su 4
6. $\mathcal{F} \cup \{\neg G\} \vdash \neg F$ \vee -MP₂ su 3 e 5
7. $\mathcal{F} \vdash \neg G \rightarrow \neg F$ \rightarrow -I su 6

(1) **Legge di Dummett.**

$$\vdash (F \rightarrow G) \vee (G \rightarrow F)$$

Soluzione:

1. $\{F\} \vdash F$ Ass
 2. $\{F\} \vdash F \vee \neg G$ \vee -I su 1
 3. $\{F\} \vdash \neg G \vee F$ \vee -Simm su 2
 4. $\{\mathcal{F}\} \vdash G \rightarrow F$ \rightarrow -Def su 3
 5. $\{F\} \vdash (G \rightarrow F) \vee (F \rightarrow G)$ \vee -I su 4
 6. $\{F\} \vdash (F \rightarrow G) \vee (G \rightarrow F)$ \vee -Simm su 5
 7. $\{\neg F\} \vdash \neg F$ Ass
 8. $\{\neg F\} \vdash \neg F \vee G$ \vee -I su 7
 9. $\{\neg F\} \vdash F \rightarrow G$ \rightarrow -Def su 8
 10. $\{\neg F\} \vdash (F \rightarrow G) \vee (G \rightarrow F)$ \vee -I su 9
 11. $\emptyset \vdash (F \rightarrow G) \vee (G \rightarrow F)$ Casi su 6 e 10
4. Dare una dimostrazione formale del fatto che $F \wedge (F \vee G)$ e $F \vee (F \wedge G)$ sono dimostrabilmente equivalenti.
5. Provare che $F \rightarrow (G \rightarrow H)$ e $F \wedge G \rightarrow H$ sono dimostrabilmente equivalenti.

1.5 Il Principio di Induzione

Principio di induzione (semplice)

Si tratta di uno *schema di assiomi* (uno per ogni proprietà P) per l'aritmetica del primo ordine. Scritto nel linguaggio (formale) del prim'ordine, è una collezione di formule che vengono assunte come *vere* nel modello dei numeri naturali (si vedano le lezioni successive del corso).

$$P(0) \wedge \forall n(P(n) \rightarrow P(n+1)) \rightarrow \forall nP(n).$$

In pratica, il principio di induzione dice che se si vuole dimostrare che una certa proprietà P vale per tutti i numeri naturali, basta verificare la proprietà P per lo 0 e poi controllare che, *per ogni* $n \in \mathbb{N}$, se la proprietà vale per n allora vale anche per $n+1$.

La verifica del fatto che P valga per 0 si dice *base* dell'induzione, mentre la dimostrazione dell'implicazione si dice *passo induttivo*. Si noti che per dimostrare il passo induttivo si *assume* innanzitutto che la proprietà P valga per n : tale ipotesi aggiuntiva viene detta *ipotesi induttiva*. Fatta quest'assunzione si cerca di concludere che P vale anche per $n+1$.

In pratica, una dimostrazione per induzione si divide in quattro passi:

1. Specificare che si darà una *dimostrazione per induzione* e su quale parametro⁸.
2. Passo base: verificare che la proprietà P vale per 0.
3. Specificare l'ipotesi induttiva $P(n)$.
4. Dall'ipotesi induttiva (più le eventuali ipotesi del teorema) dimostrare che P vale anche per $n + 1$.

Osservazione. Nella dimostrazione del passo induttivo, è importante controllare che la dimostrazione fatta sia vera *per tutti gli* n , e non solo da un certo punto in poi.

Una “variante” del principio di induzione che risulta molto utile quando si deve dimostrare che una certa proprietà vale da k in poi, è quella che si ottiene considerando come base dell'induzione (al posto dello 0) il numero k . Il passo induttivo deve essere allora verificato per $n \geq k$. Il principio di induzione⁹ si scrive allora:

$$P(k) \wedge \forall n \geq k (P(n) \rightarrow P(n + 1)) \rightarrow \forall n \geq k P(n).$$

Utilizzando questa stessa formulazione, si vede che è possibile dimostrare che P vale per tutti i numeri naturali verificando un certo numero k di casi iniziali e applicando da k in poi il principio di induzione.

Esercizi.

1. Dimostrare che per ogni $n \in \mathbb{N}$ si ha

$$(a) \quad 1 + 2 + \dots + n = \frac{n(n+1)}{2}$$

$$(b) \quad \frac{1}{2} + \frac{1}{2 \cdot 3} + \frac{1}{3 \cdot 4} + \dots + \frac{1}{n(n+1)} = \frac{n}{n+1}$$

$$(c) \quad 1 + 4 + 9 + \dots + n^2 = \frac{n(n+1)(2n+1)}{6}$$

$$(d) \quad 2 + 4 + \dots + 2n = n(n + 1)$$

$$(e) \quad 2 + 6 + 12 + \dots + (n^2 - n) = \frac{n^3 - n}{3}$$

$$(f) \quad 1 \cdot 2 + 2 \cdot 3 + 3 \cdot 4 + \dots + n(n + 1) = \frac{n(n+1)(n+2)}{3}$$

$$(g) \quad 1 + r + r^2 + \dots + r^n = \frac{r^{n+1} - 1}{r - 1}$$

2. Dimostrare che per ogni $n \in \mathbb{N}$, $n^3 - n$ è multiplo di 3.

⁸Per esempio si potrebbe iniziare la dimostrazione con “Per induzione su n ”.

⁹Si osservi che questa formulazione è riconducibile a quella data in precedenza: basta considerare la formula $P'(n) = P(n + k)$ al posto di $P(n)$ e applicare a P' il principio di induzione semplice visto sopra.

3. Dimostrare che per ogni $n \in \mathbb{N}$, $n^3 + 3n^2 + 2n$ è divisibile per 6.
4. Dimostrare che per ogni $n \in \mathbb{N}$, $n^5 + 4n + 10$ è divisibile per 5.
5. Dimostrare che per ogni $n \geq 3$, $(n + 1)^2 < 2n^2$.
6. Dimostrare che per ogni $n > 0$, 2^n divide $(n + 1)(n + 2) \cdots (2n)$.
7. Dimostrare che la somma degli angoli interni di un poligono convesso con n lati è $\pi(n - 2)$.
8. Dimostrare che ci si sono 2^n sottoinsiemi di un insieme con n elementi.
9. Dimostrare che ci sono $n!$ permutazioni di un insieme con n elementi.

Ci sono funzioni numeriche che vengono definite per *ricorsione*, cioè mediante uno schema del tipo

$$\begin{cases} f(0) = k \\ f(n + 1) = h(n, f(n)) \quad n \geq 0, \end{cases}$$

dove k è un numero naturale e h è una funzione data. Per esempio si può definire la seguente funzione

$$\begin{cases} f(0) = 1 \\ f(n + 1) = 2f(n) \quad n \geq 0. \end{cases}$$

Quando una funzione è definita per ricorsione, si può facilmente dimostrare per induzione che *i suoi valori* soddisfano una certa proprietà. Nell'esempio precedente si può per esempio dimostrare che $f(n) = 2^n$.

Esercizi.

1. Data f definita da

$$\begin{cases} f(0) = 1 \\ f(n + 1) = nf(n) + f(n) \quad n \geq 0, \end{cases}$$

dimostrare che $f(n) = n!$.

2. La Torre di Hanoi. Ci sono tre aste verticali; all'inizio su di una sono infilzati n dischi con un buco in mezzo, di raggio decrescente dal basso verso l'alto. Bisogna spostare la pila in un'altra asta, muovendo un disco alla volta da una pila e infilzandolo in un'altra, servendosi

anche della terza asta come passaggio. La condizione è che in nessun momento su nessuna pila ci sia un disco al di sotto del quale ce n'è uno di raggio minore.

Detta g la funzione tale che $g =$ “numero di mosse necessario per spostare una pila di n dischi”, trovare una definizione ricorsiva di g e dimostrare poi che $g(n) = 2^n - 1$.

Le funzioni definite ricorsivamente possono anche essere funzioni dai numeri naturali in un insieme di oggetti qualsiasi.

Un esempio è l' \wedge generalizzato (o il \vee generalizzato) visto a lezione, che può essere definito mediante una funzione dai numeri naturali (che corrisponderanno al numero di formule congiunte) nell'insieme delle formule, ovvero la funzione tale che $f(n) = \bigwedge_{i=1}^n F_i$, dove $F_1, F_2, \dots, F_n, \dots$ sono formule arbitrarie. Tale definizione può essere data direttamente (come a lezione) o con lo schema ricorsivo seguente:

$$\begin{cases} \bigwedge_{i=1}^1 F_i = F_1 \\ \bigwedge_{i=1}^{n+1} F_i = (\bigwedge_{i=1}^n F_i) \wedge F_{n+1} \quad n \geq 1. \end{cases}$$

Analogamente, la funzione $g(n) = \bigvee_{i=1}^n F_i$ può essere definita da:

$$\begin{cases} \bigvee_{i=1}^1 F_i = F_1 \\ \bigvee_{i=1}^{n+1} F_i = (\bigvee_{i=1}^n F_i) \vee F_{n+1} \quad n \geq 1. \end{cases}$$

Anche in questo caso, quando si vuole dimostrare qualcosa per i valori di queste funzioni, lo si può fare per induzione sul parametro n (sfruttando per la base e per il passo induttivo la definizione ricorsiva data).

Esercizi.

1. Date le formule F_1, F_2, \dots e G_1, G_2, \dots , provare le equivalenze logiche seguenti:

$$\begin{aligned} \text{(a)} \quad & (\bigwedge_{i=1}^n F_i) \vee (\bigwedge_{j=1}^m G_j) \equiv \bigwedge_{i=1}^n (\bigwedge_{j=1}^m (F_i \vee G_j)) \\ \text{(b)} \quad & (\bigvee_{i=1}^n F_i) \wedge (\bigvee_{j=1}^m G_j) \equiv \bigvee_{i=1}^n (\bigvee_{j=1}^m (F_i \wedge G_j)). \end{aligned}$$

2. Si definisca la funzione $f(n) = \sum_{i=1}^n a_i$ ponendo

$$\begin{cases} \sum_{i=1}^1 a_i = a_1 \\ \sum_{i=1}^{n+1} a_i = (\sum_{i=1}^n a_i) + a_{n+1} \quad n \geq 1. \end{cases}$$

Dimostrare che $\sum_{i=1}^n a_i^2 = 0$ se e solo se $a_i = 0$ per ogni $i = 1, \dots, n$.

Esistono altri due principi riguardanti i numeri naturali che sono strettamente connessi con il principio di induzione semplice.

Principio di induzione (completa)

$$\forall n[(\forall m < n P(m)) \rightarrow P(n)] \rightarrow \forall n P(n).$$

Principio del minimo

$$\exists n P(n) \rightarrow \exists n[P(n) \wedge \forall m < n (\neg P(m))].$$

Vedremo quando parleremo di logica predicativa del primo ordine che questi tre principi sono equivalenti (ovvero che le formule del prim'ordine che le definiscono sono equivalenti).

Per ora consideriamo solo il principio di induzione completa: esso permette di mostrare che una certa proprietà P vale per tutti i numeri naturali facendo vedere che per ogni $n \in \mathbb{N}$, se P vale per tutti i numeri (strettamente) più piccoli di n , allora vale anche per n . Dunque la dimostrazione viene eseguita assumendo l'ipotesi aggiuntiva (che anche in questo caso verrà chiamata *ipotesi induttiva*) che la proprietà P valga per ogni $m < n$. In realtà, pur non essendo strettamente necessario, è sempre buona norma verificare a parte il passo base ($n = 0$).

Questo è il principio che si usa quando si vuole fare una dimostrazione *per induzione sulla complessità di una formula*, ovvero per induzione sul grado (= altezza dell'albero $- 1$) di tale formula.

La ragione di ciò è che se F ha grado $n + 1$, non è detto che *entrambe* le sue sottoformule principali abbiano grado n : l'unica cosa certa è che tutte le sottoformule proprie di F hanno grado strettamente minore di essa.

Dunque quando si vuole dimostrare per induzione sulla complessità (= grado) di una formula che una proprietà P vale per tutte le formule, bisogna:

1. verificare che P valga per le formule di grado 0, ovvero per le formule atomiche;
2. data F di grado n , si assume l'ipotesi induttiva che P valga per tutte le formule di grado strettamente minore di n , e da questo si cerca di concludere che P vale anche per F . Poiché abbiamo definito le formule utilizzando \neg e \wedge , bisogna controllare che P valga per H quando H è della forma $H = \neg F$ oppure $H = F \wedge G$, supposto che P valga per F e G (che hanno grado strettamente minore di H).

Osservazione. Si osservi che se un insieme C è definito induttivamente, è sempre possibile associare ai suoi elementi un grado, e quindi quando si vuole dimostrare che una proprietà vale per tutti i suoi elementi si può sempre fare un'induzione sul grado. Questo giustifica anche il nome di *definizione induttiva* che avevamo dato.

Esercizi.

1. **Teorema di sostituzione (sintattico).** Supponiamo che F e G siano due formule dimostrabilmente equivalenti. Sia H una formula che contiene F come sottoformula, e sia H' la formula ottenuta da H rimpiazzando qualche occorrenza di F con G . Dimostrare che H e H' sono dimostrabilmente equivalenti.
2. Dimostrare che per ogni formula proposizionale, la sua lunghezza (come stringa di simboli) è strettamente minore del suo grado (= altezza dell'albero sintattico $- 1$).

1.6 Forme normali

Definizione 8. Una letterale è una formula atomica (letterale positivo) o la negazione di una formula atomica (letterale negativo).

Due letterali che siano uno la negazione dell'altro (ovvero siano del tipo A e $\neg A$ per qualche formula atomica A) si dicono associati.

Definizione 9. Una formula F è in Forma Normale Congiuntiva (CNF) se è la congiunzione di disgiunzioni di letterali, ovvero

$$F = \bigwedge_{i=1}^n \left(\bigvee_{j=1}^m L_{ij} \right),$$

dove ogni L_{ij} è un letterale (positivo o negativo).

Definizione 10. Una formula F è in Forma Normale Disgiuntiva (DNF) se è la disgiunzione di congiunzioni di letterali, ovvero

$$F = \bigvee_{i=1}^n \left(\bigwedge_{j=1}^m L_{ij} \right),$$

dove ogni L_{ij} è un letterale (positivo o negativo).

Lemma 4. Sia F una formula in CNF e G una formula in DNF. Allora $\neg F$ è (logicamente) equivalente ad una formula in DNF e $\neg G$ è (logicamente) equivalente ad una formula in CNF.

Dimostrazione. Sia

$$F = \bigwedge_{i=1}^n \left(\bigvee_{j=1}^m L_{ij} \right),$$

dove gli L_{ij} sono letterali. Allora

$$\neg F = \neg \bigwedge_{i=1}^n \left(\bigvee_{j=1}^m L_{ij} \right) \equiv \bigvee_{i=1}^n \neg \left(\bigvee_{j=1}^m L_{ij} \right)$$

per la legge di De Morgan generalizzata. Inoltre, per il teorema di sostituzione semantico e De Morgan generalizzato, si ha

$$\neg F \equiv \bigvee_{i=1}^n \left(\bigwedge_{j=1}^m \neg L_{ij} \right),$$

e poiché la negazione di un letterale $\neg L_{ij}$ è sempre equivalente ad un letterale L'_{ij} , di nuovo per il teorema di sostituzione semantico si ha che

$$\neg F \equiv \bigvee_{i=1}^n \left(\bigwedge_{j=1}^m L'_{ij} \right).$$

Analogamente si prova la seconda parte del Lemma. \square

Teorema 5. *Ogni formula F è (logicamente) equivalente a una formula F_1 in CNF e ad una formula F_2 in DNF, entrambe con le stesse lettere di F .*

Dimostrazione. Per induzione sulla complessità della formula F . Se F è atomica allora è già sia in CNF che in DNF, quindi possiamo considerare $F_1 = F_2 = F$.

Sia ora $F = \neg G$. Poiché G ha grado minore di F , per ipotesi induttiva esistono G_1 in CNF e G_2 in DNF tali che $G \equiv G_1 \equiv G_2$. Allora $F \equiv \neg G_2 \equiv \neg G_1$, e per il Lemma precedente $\neg G_2$ è equivalente ad una formula F_1 in CNF e $\neg G_1$ è equivalente ad una formula F_2 in DNF: quindi si ha anche $F \equiv F_1 \equiv F_2$.

Infine sia $F = G \wedge H$ e siano G_1 e H_1 formule in CNF e G_2 e H_2 formule in DNF tali che $G \equiv G_1 \equiv G_2$ e $H \equiv H_1 \equiv H_2$ (tali formule esistono per ipotesi induttiva grazie al fatto che G e H hanno grado inferiore a quello di F). È facile verificare che $F_1 = G_1 \wedge H_1$ è ancora in CNF e $F \equiv F_1$. Inoltre possiamo assumere che $G_2 = \bigvee_{i=1}^n M_i$ e $H_2 = \bigvee_{j=1}^m N_j$, dove ciascun M_i e N_j è congiunzione di letterali. Allora si ha che

$$F \equiv G_2 \wedge H_2 = \left(\bigvee_{i=1}^n M_i \right) \wedge \left(\bigvee_{j=1}^m N_j \right) \equiv \bigvee_{i=1}^n \left(\bigvee_{j=1}^m (M_i \wedge N_j) \right) = F_2$$

per la regola di distributività generalizzata, e F_2 è in DNF. \square

La formula F_1 del teorema precedente verrà detta *forma normale congiuntiva di F* (o, brevemente, $\text{CNF}(F)$), mentre F_2 verrà detta *forma normale disgiuntiva di F* (in simboli, $\text{DNF}(F)$).

Osservazione. La forma normale congiuntiva (o disgiuntiva) di una formula F non è determinata in maniera univoca. Infatti se F_1 è in CNF e $F_1 \equiv F$, considerando la tautologia $T = A \vee \neg A$, anche $F_1 \wedge T$ è in CNF e inoltre è equivalente ad F . Tuttavia, se F_1 e F_1^* sono due forme normali congiuntive di F , allora sono tra di loro equivalenti.

Analogamente per le forme normali disgiuntive: se F_2 è in DNF e $F_2 \equiv F$, considerata la contraddizione $\perp = A \wedge \neg A$, si ha che anche $F_2 \vee \perp$ è in DNF ed è equivalente ad F .

Dunque, $\text{CNF}(F)$ e $\text{DNF}(F)$ sono determinate *a meno di equivalenze logiche*.

Data (la colonna finale di) una tavola di verità, è possibile ricostruire una formula che corrisponderà a tale tavola. In realtà, esistono due algoritmi che permettono di costruire una tale formula che sia, rispettivamente, in DNF o in CNF.

Algoritmo 1 (per le formule DNF):

1. Per ogni riga corrispondente ad un 1 nella colonna finale della tavola di verità si considera l'assegnamento corrispondente;
2. ad ogni formula atomica che compare, si associa un letterale: in forma positiva se l'assegnamento dà valore 1 alla formula atomica in questione, in forma negativa se invece dà valore 0;
3. per ogni assegnamento preso in considerazione secondo il primo passo, si costruisce la congiunzione dei letterali ottenuti al secondo passo;
4. si fa la disgiunzione delle formule ottenute al passo precedente.

Algoritmo 2 (per le formule CNF):

1. Per ogni riga corrispondente ad uno 0 nella colonna finale della tavola di verità si considera l'assegnamento corrispondente;
2. ad ogni formula atomica si associa un letterale: in forma positiva se l'assegnamento dà valore 0 alla formula atomica in questione, in forma negativa se invece dà valore 1;
3. per ogni assegnamento preso in considerazione secondo il primo passo, si costruisce la disgiunzione dei letterali ottenuti al secondo passo;
4. si fa la congiunzione delle formule ottenute al passo precedente.

Esercizi.

1. Trovare una formula F in CNF che ha la seguente tavola di verità:

A	B	C	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

2. Trovare una formula in DNF che ha la tavola di verità dell'esercizio precedente.

3. Trovare una formula F in CNF che ha la seguente tavola di verità:

A	B	C	F
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

4. Trovare una formula in DNF che ha la tavola di verità dell'esercizio precedente.

Data una formula F , gli algoritmi 1 e 2 permettono di ricavare due formule (rispettivamente, in CNF e in DNF) ad essa equivalenti: basta calcolare la tavola di verità di F e da questa ricavare le due formule con i rispettivi algoritmi.

Esercizi.

1. Scrivere la forma normale congiuntiva e disgiuntiva, usando le tavole di verità, delle seguenti proposizioni:

$$(A \vee B \rightarrow C) \wedge \neg A \wedge \neg C$$

$$\neg A \rightarrow \neg(B \rightarrow A)$$

$$(\neg(A \rightarrow B) \vee \neg B) \rightarrow A$$

Introduciamo ora altri due algoritmi che permettono di mettere una formula F in forma normale (coniuntiva o disgiuntiva), che si basano sulla validità delle leggi di Doppia Negazione, De Morgan e distributività di \wedge e \vee .

Algoritmo CNF

1. Sostituire ogni sottoformula del tipo $F \rightarrow G$ (se esiste) con $\neg F \vee G$, e tutte le sottoformule del tipo $F \leftrightarrow G$ (se ce ne sono) con $(\neg F \vee G) \wedge (\neg G \vee F)$;
2. utilizzando le leggi di De Morgan e la Doppia Negazione, portare tutti i \neg verso l'interno in modo da ottenere solo congiunzioni e disgiunzioni di letterali, ovvero sostituire le sottoformule del tipo $\neg\neg G$ con G , quelle del tipo $\neg(F \wedge G)$ con $\neg F \vee \neg G$ e quelle del tipo $\neg(F \vee G)$ con $\neg F \wedge \neg G$;
3. applicare la legge di distributività del \vee rispetto all' \wedge dove possibile, ovvero sostituire le formule del tipo $G \vee (H \wedge K)$ o $(H \wedge K) \vee G$ con $(G \vee H) \wedge (G \vee K)$.

Algoritmo DNF

1. Come nell'algoritmo CNF;
2. come nell'algoritmo CNF;
3. applicare la legge di distributività dell' \wedge rispetto al \vee dove possibile, ovvero sostituire le formule del tipo $G \wedge (H \vee K)$ o $(H \vee K) \wedge G$ con $(G \wedge H) \vee (G \wedge K)$.

Osservazione. Gli algoritmi CNF e DNF permettono di rafforzare il teorema citato all'inizio, nella seguente maniera: data una formula F esistono due formule F_1 ed F_2 in, rispettivamente, CNF e DNF che sono *dimostrabilmente equivalenti* ad F (per la correttezza, questo implica che sono anche logicamente equivalenti a F). Per verificare ciò basta considerare gli algoritmi passo per passo. Ad ogni passo semplicemente vengono sostituite alcune sottoformule con altre formule, e in ciascuno di questi casi, si può provare che esse sono dimostrabilmente equivalenti. Per esempio $F \rightarrow G \dashv\vdash \neg F \vee G$ e $F \leftrightarrow G \dashv\vdash (\neg F \vee G) \wedge (\neg G \vee F)$ (e così via). Dunque, applicando il teorema di sostituzione sintattico, si ottiene che il risultato voluto.

Teorema 6. *Sia F una formula in CNF. Allora essa è una tautologia sse in ogni suo congiunto compare una lettera proposizionale che vi occorre sia in forma positiva che in forma negativa.*

Sia invece G una formula in DNF. Allora essa è insoddisfacibile se e solo se in ogni suo disgiunto compare una lettera proposizionale che vi occorre sia in forma positiva che in forma negativa.

Dimostrazione. Sia $F = \bigwedge_{i=1}^n \left(\bigvee_{j=1}^m L_{ij} \right)$ una formula in CNF, dove L_{ij} sono letterali. Se per ogni i esistono j e j' tali che L_{ij} e $L_{ij'}$ sono letterali associati, allora per *ogni* assegnamento \mathcal{A} si deve avere $\mathcal{A} \models L_{ij}$ oppure $\mathcal{A} \models L_{ij'}$. Dunque, in ogni caso, $\mathcal{A} \models \bigvee_{j=1}^m L_{ij}$ per ogni i , ovvero $\mathcal{A} \models F$. Essendo \mathcal{A} arbitrario, F è una tautologia.

Viceversa, sia i tale che non vi siano letterali associati tra L_{i1}, \dots, L_{im} , e siano A_1, \dots, A_k le lettere proposizionali che compaiono in F . Sia \mathcal{A} un assegnamento definito nella maniera seguente:

$$\mathcal{A}(A_l) = \begin{cases} 0 & \text{se } L_{ij} \neq A_l \text{ e } L_{ij} \neq \neg A_l \text{ per ogni } j \\ 1 & \text{se } L_{ij} = \neg A_l \text{ per qualche } j \\ 0 & \text{se } L_{ij} = A_l \text{ per qualche } j \end{cases}$$

per ogni $1 \leq l \leq k$. Allora $\mathcal{A} \not\models \bigvee_{j=1}^m L_{ij}$ e quindi $\mathcal{A} \not\models F$ (cioè F non è una tautologia).

Sia ora $G = \bigvee_{i=1}^n \left(\bigwedge_{j=1}^m L_{ij} \right)$ una formula in DNF e supponiamo che per ogni i esistano j e j' tali che L_{ij} e $L_{ij'}$ sono letterali associati. Allora per *ogni* assegnamento \mathcal{A} si ha che $\mathcal{A} \not\models L_{ij} \wedge L_{ij'}$ e dunque $\mathcal{A} \not\models \bigwedge_{j=1}^m L_{ij}$ per ogni i , cioè G è insoddisfacibile.

Viceversa, siano A_1, \dots, A_k le lettere proposizionali che compaiono in G e sia i tale che non vi siano letterali associati tra L_{i1}, \dots, L_{im} . Definiamo l'assegnamento \mathcal{A} ponendo

$$\mathcal{A}(A_l) = \begin{cases} 0 & \text{se } L_{ij} \neq A_l \text{ e } L_{ij} \neq \neg A_l \text{ per ogni } j \\ 1 & \text{se } L_{ij} = A_l \text{ per qualche } j \\ 0 & \text{se } L_{ij} = \neg A_l \text{ per qualche } j \end{cases}$$

per ogni $1 \leq l \leq k$. Allora $\mathcal{A} \models \bigwedge_{j=1}^m L_{ij}$ e quindi $\mathcal{A} \models G$: dunque G è soddisfacibile. \square

Osservazione. Questo Teorema dimostra anche che il *problema della validità per le formule in CNF* e il *problema della soddisfacibilità per le formule in DNF* sono risolvibili (ovvero ammettono un algoritmo che lavora) in tempo polinomiale. Tuttavia queste considerazioni non risolvono il problema della validità (o della soddisfacibilità) *per formule arbitrarie*: infatti non si conoscono algoritmi efficienti per, data una formula qualsiasi F , trovare una formula ad essa (logicamente) equivalente che sia in CNF (o in DNF). La prima coppia di algoritmi non è efficiente perché richiede di calcolare la tavola di verità di F , e come già osservato le tavole di verità non si possono in generale calcolare in tempo polinomiale. Gli algoritmi CNF e DNF, d'altra parte, non lavorano in tempo polinomiale per via del numero di applicazioni delle regole di distributività necessarie. Infatti, se $F = (A_1 \wedge B_1) \vee (A_2 \wedge B_2) \vee \dots \vee (A_n \wedge B_n)$ e stiamo applicando l'algoritmo CNF, bisognerà applicare la regola di distributività di \wedge su \vee per 2^{n-1} volte prima di ottenere una formula in CNF, mentre la lunghezza di F (in numero di simboli

che vi occorrono) è $n \cdot 5 + (n - 1) = 6n - 1$: ma $6n - 1 < 2^{n-1}$ per $n > 6$.

Inoltre, se F è in DNF e non è insoddisfacibile, dalla dimostrazione precedente si possono ricavare facilmente degli assegnamenti che la modellano.

Per ogni disgiunto che non contenga nessuna lettera proposizionale in forma sia positiva che negativa, si assegna valore 1 alle lettere proposizionali che vi occorrono in forma positiva e valore 0 a quelle che vi occorrono in forma negativa; infine si assegna un valore arbitrario alle lettere proposizionali che compaiono altrove nella formula, ma non nel disgiunto in questione. Quelli così ottenuti sono tutti (e soli) i modelli per la formula F .

Un analogo algoritmo permette di ricavare dalla corrispondente formula in CNF i controesempi al fatto che una certa formula F sia una tautologia, ovvero consente di ottenere (tutti e soli) gli assegnamenti che non modellano F .

Si considerano i congiunti in cui non compare nessuna lettera proposizionale sia in forma positiva che in forma negata: alle lettere proposizionali che occorrono in un tale disgiunto, si assegna 0 se appaiono in forma positiva e 1 se vi appaiono in forma negata (come prima, alle lettere proposizionali che occorrono altrove nella formula ma non nel congiunto in questione, si può assegnare un valore arbitrario, ottenendo così assegnamenti diversi). Gli assegnamenti così ottenuti sono i controesempi cercati.

Esercizi.

1. Scrivere la forma normale congiuntiva e disgiuntiva delle seguenti formule utilizzando gli algoritmi CNF e DNF:

$$(A \vee B) \rightarrow \neg(A \rightarrow (B \rightarrow C))$$

$$(A \vee B) \rightarrow \neg(A \wedge (B \rightarrow C))$$

$$A \rightarrow (\neg B \vee A \rightarrow (C \rightarrow A))$$

$$(A \rightarrow B) \rightarrow (C \rightarrow \neg A)$$

$$A \vee B \rightarrow \neg A \vee B$$

$$A \vee (B \wedge C) \rightarrow (\neg C \rightarrow A)$$

2. Per ciascuna delle formule seguenti, trovare una formula in CNF equivalente a quella data. Verificare se sono tautologie e, in caso contrario, trovare dei controesempi:

$$(\neg A \rightarrow B) \vee ((A \wedge \neg C) \leftrightarrow B)$$

$$(A \rightarrow B \wedge \neg B) \rightarrow \neg A$$

$$(A \rightarrow B) \wedge (A \rightarrow \neg B)$$

$$(A \rightarrow (B \vee C)) \vee (C \rightarrow \neg A)$$

$$A \wedge (\neg A \vee B) \rightarrow B$$

$$((A \rightarrow B) \wedge C) \vee (A \wedge D)$$

$$(A \leftrightarrow B) \leftrightarrow C$$

$$(A \rightarrow B \vee C) \vee (C \rightarrow \neg A)$$

$$(\neg A \wedge \neg B \wedge C) \vee (\neg A \wedge \neg C) \vee (B \wedge C) \vee A$$

3. Scrivere $\neg A \vee B \rightarrow \neg A \wedge B$ in DNF e ricavarne i modelli.

4. Porre la seguente formula in CNF:

$$(A \wedge \neg B) \vee C \rightarrow \neg(\neg D \rightarrow E \wedge \neg F)$$

5. Porre la seguente formula in CNF e in DNF:

$$\neg(A \wedge B \rightarrow C) \vee (D \rightarrow \neg B)$$

6. Dire, utilizzando la sua forma normale disgiuntiva, se la seguente formula è insoddisfacibile e, in caso contrario, ricavarne i modelli:

$$(A \rightarrow B) \rightarrow \neg(\neg(A \rightarrow A \vee B))$$

Algoritmo di Horn

Esiste un'altra classe ristretta di formule che ammette un algoritmo che risolve il problema della soddisfacibilità (per tali formule) in un tempo polinomiale.

Definizione 11. Una formula F si dice formula di Horn se è in CNF e ogni congiunto (che consiste di una disgiunzione) contiene al più un letterale positivo.

F si dice formula di Horn di base se è una formula di Horn costituita da un unico congiunto. Dunque ogni formula di Horn è congiunzione di formule di Horn di base.

Se F è una formula di Horn, ogni congiunto può essere sostituito da una formula ad esso (logicamente e dimostrabilmente) equivalente nella seguente maniera:

1. se il disgiunto contiene sia letterali negativi, sia il letterale positivo (unico), allora può essere sostituito da un'implicazione che coinvolge solo letterali positivi. Per esempio, se il disgiunto è $(\neg A \vee \neg B \vee C)$, viene sostituito dall'equivalente $(A \wedge B) \rightarrow C$;
2. se il disgiunto non contiene nessun letterale positivo, allora può essere sostituito da un'implicazione che coinvolge una contraddizione. Per esempio, se il disgiunto è $(\neg B \vee \neg D)$ e \perp è una contraddizione, allora il disgiunto può essere sostituito da $(B \wedge D) \rightarrow \perp$;

3. se il disgiunto non contiene nessun letterale negativo (ovvero consiste di un unico letterale positivo), può essere sostituito da un'implicazione che coinvolge una tautologia. Per esempio, se il disgiunto è A e T è una tautologia, può essere sostituito con $T \rightarrow A$.

In questa maniera ogni formula di Horn può essere facilmente trasformata in una congiunzione di implicazioni. D'ora in poi, con il termine, "formula di Horn" (rispettivamente, formula di Horn di base) intenderemo una formula di Horn (nel senso della definizione precedente) che sia stata posta proprio in questa forma.

Supponiamo ora che ci venga fornita una formula di Horn H e si voglia decidere se H sia soddisfacibile o no: un tale problema di decisione verrà detto *problema della soddisfacibilità di Horn*. Presentiamo un algoritmo che è in grado di decidere tale problema in tempo polinomiale.

Algoritmo HORN

Data una formula di Horn H (congiunzione di implicazioni), facciamo una lista di tutte le formule atomiche occorrenti in H . L'algoritmo comprende tre passi.

1. Segnare ogni formula A della lista che sia in una sottoformula del tipo $T \rightarrow A$;
2. se esiste una sottoformula del tipo $(A_1 \wedge \dots \wedge A_m) \rightarrow C$, dove ogni A_i è stato segnato e C non è ancora segnato, allora si segna C . Si ripete questo passo fino a che non vi sono più sottoformule di questo tipo e poi si procede al passo successivo;
3. si considerano le sottoformule del tipo $(A_1 \wedge \dots \wedge A_m) \rightarrow \perp$. Se esiste una tale sottoformula in cui *ogni* A_i è stato segnato, allora si conclude "No, H non è soddisfacibile". Altrimenti si conclude "Sì, H è soddisfacibile".

Si può dimostrare che l'algoritmo HORN è corretto e completo, cioè che conclude "Sì, H è soddisfacibile" se e solo se H è soddisfacibile.

Teorema 7. *L'algoritmo di Horn è corretto e completo.*

Dimostrazione. Sia H una formula di Horn e siano A_1, \dots, A_n le lettere proposizionali che occorrono in H . Supponiamo che H sia soddisfacibile e sia \mathcal{A} un assegnamento tale che $\mathcal{A} \models H$. In particolare, $\mathcal{A}(B) = 1$ per ogni formula di Horn di base che compone H (ovvero per ogni congiunto di H). Supponiamo di applicare l'algoritmo di Horn alla formula H : dimostreremo innanzitutto che se A_i è stata segnata durante l'applicazione di tale algoritmo allora $\mathcal{A}(A_i) = 1$. Se A_i è stata segnata poiché compare in una sottoformula

di H del tipo $T \rightarrow A_i$ allora, essendo T una tautologia, dal fatto che $\mathcal{A} \models T \rightarrow A_i$ si deve avere che $\mathcal{A}(A_i) = 1$. Se invece A_i è stata segnata poiché compare in una formula del tipo $C_1 \wedge \dots \wedge C_m \rightarrow A_i$ dove C_1, \dots, C_m sono state segnate (ovvero $\mathcal{A}(C_j) = 1$ per ogni $1 \leq j \leq m$), allora da $\mathcal{A} \models C_1 \wedge \dots \wedge C_m \rightarrow A_i$ segue $\mathcal{A}(A_i) = 1$. Supponiamo ora per assurdo che l'algoritmo termini concludendo "No, H non è soddisfacibile". Allora H contiene una formula di Horn di base del tipo $C_1 \wedge \dots \wedge C_m \rightarrow \perp$ dove C_1, \dots, C_m sono state tutte segnate. Per quanto dimostrato prima, si deve avere anche $\mathcal{A}(C_j) = 1$ per ogni $1 \leq j \leq m$ e dunque $\mathcal{A} \not\models C_1 \wedge \dots \wedge C_m \rightarrow \perp$ (siccome $\mathcal{A} \models C_1 \wedge \dots \wedge C_m$ e \perp è una contraddizione), contraddizione!

Viceversa, supponiamo che l'algoritmo termini rispondendo "Sì, H è soddisfacibile" e sia \mathcal{A} l'assegnamento definito ponendo $\mathcal{A}(A_i) = 1$ se A_i è stato segnato e $\mathcal{A}(A_i) = 0$ altrimenti. Sia B una formula di Horn di base che compone H . È sufficiente dimostrare che per ogni tale B si ha $\mathcal{A} \models B$. Se B è del tipo $T \rightarrow A_i$ allora A_i è stata segnata e dunque $\mathcal{A} \models B$. Se B è del tipo $C_1 \wedge \dots \wedge C_m \rightarrow A_i$ e ciascun C_j è stato segnato allora anche A_i è stato segnato e dunque $\mathcal{A} \models B$. Se invece esiste un C_j che non è stato segnato, allora $\mathcal{A} \not\models C_1 \wedge \dots \wedge C_m$ e quindi nuovamente $\mathcal{A} \models B$. Infine, se B è del tipo $C_1 \wedge \dots \wedge C_m \rightarrow \perp$, poiché l'output dell'algoritmo è stato che H è soddisfacibile deve esistere qualche j tale che C_j non è stato segnato: dunque $\mathcal{A} \not\models C_1 \wedge \dots \wedge C_m$, da cui $\mathcal{A} \models B$. \square

Dalla dimostrazione segue che se H risulta soddisfacibile, allora si può costruire un assegnamento che testimonia ciò semplicemente dando valore 1 alle formule atomiche della lista che sono state segnate e 0 alle altre.

Inoltre, si può dimostrare che l'algoritmo termina in meno di n^2 passi (dove $n > 2$ è la lunghezza, come numero di simboli, della formula H data come input¹⁰ e per "numero di passi" intendiamo il numero di volte che deve essere letta la formula H). Infatti, è necessaria una lettura per fare una lista delle formule atomiche che compaiono in H ed un'altra lettura per segnare quelle che compaiono in formule del tipo $T \rightarrow A$ (ovvero per eseguire il primo passo dell'algoritmo). Poi bisogna leggere H ogni volta che si deve applicare il secondo passo: poiché H contiene n simboli (e quindi al più n formule atomiche), vi sono non più di n formule che possono essere segnate, dunque il secondo passo può essere eseguito al più n volte. Infine è necessaria un'ulteriore lettura di H per eseguire il terzo passo dell'algoritmo. In conclusione, sono necessarie $1 + 1 + n + 1$ letture di H affinché l'algoritmo di Horn termini, e $n + 3 < n^2$ per $n > 2$.

Esercizi.

¹⁰Se $n \leq 2$ allora la formula in questione è un letterale, ed è sempre soddisfacibile.

1. Determinare se le seguenti formule di Horn sono soddisfacibili e, in caso positivo, trovare un assegnamento che modelli la formula:

$$A_1 \wedge A_2 \wedge (\neg A_1 \vee \neg A_2 \vee \neg A_3 \vee A_4) \wedge (\neg A_1 \vee \neg A_2 \vee \neg A_4 \vee A_5) \wedge \\ \wedge (\neg A_1 \vee \neg A_2 \vee \neg A_3 \vee \neg A_4 \vee A_6) \wedge (\neg A_5 \vee \neg A_6 \vee A_7) \wedge (\neg A_2 \vee A_3) \wedge (\neg A_7)$$

$$(T \rightarrow A_1) \wedge (T \rightarrow A_2) \wedge (A_1 \wedge A_2 \wedge A_4 \rightarrow A_3) \wedge (A_1 \wedge A_5 \wedge A_6 \rightarrow \perp) \wedge \\ \wedge (A_2 \wedge A_7 \rightarrow A_5) \wedge (A_1 \wedge A_3 \wedge A_5 \rightarrow A_7) \wedge (A_2 \rightarrow A_4) \wedge (A_4 \rightarrow A_8) \wedge \\ \wedge (A_2 \wedge A_3 \wedge A_4 \rightarrow A_9) \wedge (A_3 \wedge A_9 \rightarrow A_6) \wedge (A_6 \wedge A_7 \rightarrow A_8) \wedge (A_7 \wedge A_8 \wedge A_9 \rightarrow \perp)$$

1.7 Risoluzione

Supponiamo F sia in CNF, ovvero sia una congiunzione di disgiunzioni di letterali. Allora chiameremo ogni disgiunzione di letterali *clausola* della formula F . In realtà considereremo ogni clausola $C = L_1 \vee \dots \vee L_n$ come l'insieme dei letterali che vi occorrono, ovvero l'insieme $C = \{L_1, \dots, L_n\}$.

Proposizione 8. *Siano C e D due clausole. Se $C = D$ (come insiemi) allora $C \equiv D$ (come formule, cioè disgiunzioni di letterali).*

Dimostrazione. Sia $\{L_1, \dots, L_n\}$ l'insieme dei letterali che occorrono in C (o, equivalentemente, in D). Allora poiché $F \vee F \equiv F$ per ogni formula F , si deve avere

$$C \equiv L_1 \vee \dots \vee L_n \equiv D.$$

□

Analogamente, poiché una formula $F = C_1 \wedge \dots \wedge C_n$ in CNF è congiunzione di clausole, considereremo una tale formula come un insieme di clausole (viste a loro volta come insiemi¹¹), ovvero come l'insieme $F = \{C_1, \dots, C_n\}$.

Proposizione 9. *Siano F e G due formule in CNF. Se $F = G$ (come insiemi) allora $F \equiv G$ (come formule).*

Dimostrazione. Sia $\{C_1, \dots, C_n\}$ l'insieme delle clausole che occorrono in F (o, equivalentemente, in G). Allora poiché $H \wedge H \equiv H$ per ogni formula H , si ha

$$F \equiv C_1 \wedge \dots \wedge C_n \equiv G.$$

□

¹¹Dunque vedremo una formula in CNF come insieme di insiemi di letterali, e viceversa.

Inoltre, se F e G sono due formule in CNF e $H = F \wedge G$, allora anche H è in CNF e, vista come insieme, risulta essere $H = F \cup G$.

Definizione 12. *Siano C_1 e C_2 due clausole, e supponiamo che $A \in C_1$ e $\neg A \in C_2$ per qualche formula atomica A . Allora la clausola $R = (C_1 \setminus \{A\}) \cup (C_2 \setminus \{\neg A\})$ si dice risolvente di C_1 e C_2 .*

La risolvente di due clausole potrebbe essere l'insieme vuoto. Può essere comodo estendere in questo contesto la definizione di clausola in modo da includere la clausola vuota. La clausola vuota è insoddisfacibile, perché per nessun assegnamento può dare il valore 1 a un letterale della clausola, ed è l'unica clausola che da sola è insoddisfacibile (esercizio).

Regole di deduzione (formale) per la risoluzione

1. Sia G una qualunque formula. Sia F la sua forma normale congiuntiva ottenuta mediante l'algoritmo CNF. Allora F può essere dedotto da G .
2. Sia F una formula in CNF. Ogni clausola di F può essere dedotta da F .
3. Sia F una formula in CNF. Ogni risolvente di due clausole di F può essere dedotta da F .

Teorema 10 (Correttezza della risoluzione). *Il metodo di deduzione della risoluzione è corretto.*

Dimostrazione. È sufficiente dimostrare che ciascuna delle tre regole che si utilizzano nella risoluzione è derivabile e poi utilizzare il fatto che il calcolo delle derivazioni formali è corretto.

La prima regola è derivabile per le osservazioni fatte nella Sezione precedente, mentre la seconda segue da \wedge -Simm e \wedge -E. Siano infine $C_1 = A \vee F$ e $C_2 = \neg A \vee G$ due clausole, e sia R una loro risolvente. Vogliamo provare che $\{C_1, C_2\} \vdash R$.

1. $\{C_1, C_2\} \vdash A \vee F$ Ass
2. $\{C_1, C_2\} \vdash \neg A \vee G$ Ass
3. $\{C_1, C_2, A\} \vdash \neg A \vee G$ Mon su 2
4. $\{C_1, C_2, A\} \vdash A$ Ass
5. $\{C_1, C_2, A\} \vdash G$ \vee -Mp su 3 e 4
6. $\{C_1, C_2, A\} \vdash G \vee F$ \vee -I su 5
7. $\{C_1, C_2, A\} \vdash F \vee G$ \vee -Simm su 6
8. $\{C_1, C_2, F\} \vdash F$ Ass
9. $\{C_1, C_2, F\} \vdash F \vee G$ \vee -I su 8
10. $\{C_1, C_2\} \vdash F \vee G$ \vee -E su 1, 7 e 9

□

Vediamo ora come applicare la risoluzione per controllare se una qualunque formula proposizionale F sia soddisfacibile. Per la prima regola, possiamo assumere che F sia in CNF.

Definiamo¹² $Res^0(F) = \{C \mid C \text{ è una clausola di } F\} = \{C \mid C \in F\}$. La clausola vuota non appartiene a $Res^0(F)$. Per ogni $n > 0$, sia

$$Res^n(F) = Res^{n-1}(F) \cup \{R \mid R \text{ è la risolvente di due clausole di } Res^{n-1}(F)\}.$$

Osserviamo che per ogni $n \in \mathbb{N}$, $Res^n(F)$ è un insieme di clausole (e quindi una formula in CNF): dunque ad esso si possono applicare le seconda e la terza regola per ottenere $Res^{n+1}(F)$. Inoltre, poiché $Res^0(F) = F$ è un insieme finito, da esso possono essere derivati soltanto un numero finito di clausole utilizzando la risoluzione (questo perché ci sono solo una quantità finita di clausole che usano le stesse formule atomiche di F ¹³). Quindi esisterà un m tale che $Res^m(F) = Res^{m+1}(F)$. Indicheremo con $Res^*(F)$ un tale $Res^m(F)$. Questo è l'insieme (finito) di tutte le clausole che possono essere dedotte da F usando la risoluzione.

Proposizione 11 (Completezza della risoluzione). *Sia F una formula in CNF. F è insoddisfacibile se e solo se $\emptyset \in Res^*(F)$.*

Dimostrazione. Sia F tale che $\emptyset \in Res^*(F)$ e sia m tale che $\emptyset \in Res^m(F)$ ma $\emptyset \notin Res^{m-1}(F)$ ($m > 0$ perché $\emptyset \notin Res^0(F)$). Dunque \emptyset deve essere la risolvente di due clausole $C_1, C_2 \in Res^{m-1}(F)$, e ciò è possibile solo se C_1 e C_2 sono del tipo, rispettivamente, $\{A\}$ e $\{\neg A\}$. Ma per il teorema di correttezza per la risoluzione, questo implica che $F \models A$ e $F \models \neg A$, da cui $F \vdash A \wedge \neg A$. Quindi se F fosse soddisfacibile si dovrebbe avere che anche $A \wedge \neg A$ lo è, contraddizione!

Sia ora $F = \{C_1, \dots, C_k\}$ una formula insoddisfacibile. Chiaramente deve esistere almeno qualche C_i che non sia una tautologia (altrimenti F sarebbe a sua volta una tautologia), quindi non è restrittivo assumere che ciascuna clausola *non* sia una tautologia (se C_i è valido si ha $F \equiv C_1 \wedge \dots \wedge C_{i-1} \wedge C_{i+1} \wedge \dots \wedge C_k$, quindi si può semplicemente sopprimere C_i dalla formula F). Dimostreremo che $\emptyset \in Res^*(F)$ per induzione sul numero n di formule atomiche che occorrono in F .

Se $n = 1$ allora ogni C_i è uno tra $\{A\}$ e $\{\neg A\}$ (la clausola $\{A, \neg A\}$ è una tautologia e quindi, per l'assunzione precedente, non può essere una clausola di F). Dunque si hanno tre possibilità: se $F = \{\{A\}\}$ o $F = \{\{\neg A\}\}$ si avrebbe che F è soddisfacibile (contraddicendo la nostra ipotesi), mentre se $F = \{\{A\}, \{\neg A\}\}$ si ha chiaramente $\emptyset \in Res^1(F) = Res^*(F)$.

¹²Si osservi che la definizione seguente è una definizione ricorsiva.

¹³Per la precisione, se in F occorrono n formule atomiche, allora si possono avere solo $2n$ letterali, e quindi 2^{2n} clausole distinte.

Supponiamo ora che le formule atomiche occorrenti in F siano A_1, \dots, A_{n+1} e che $\emptyset \in Res^*(G)$ per ogni formula G insoddisfacibile che contenga al più A_1, \dots, A_n come sottoformule atomiche. Sia \overline{F}_0 la congiunzione di tutte le clausole di F che non contengano $\neg A_{n+1}$ e \overline{F}_1 la congiunzione di tutte le clausole di F che non contengano A_{n+1} . Chiaramente entrambe queste formule sono in CNF. Ogni clausola di F deve appartenere o a \overline{F}_0 oppure a \overline{F}_1 (se contenesse sia A_{n+1} che $\neg A_{n+1}$ sarebbe una tautologia), mentre il viceversa è ovvio. Dunque $\overline{F}_0 \cup \overline{F}_1 = F$ (come insiemi di clausole).

Siano ora $F_0 = \{C_i \setminus \{A_{n+1}\} \mid C_i \in \overline{F}_0\}$ e $F_1 = \{C_i \setminus \{\neg A_{n+1}\} \mid C_i \in \overline{F}_1\}$. Supponiamo ora che F_0 sia soddisfacibile e sia \mathcal{A} un assegnamento che testimonia ciò. Definendo $\mathcal{A}'(A_j) = \mathcal{A}(A_j)$ per $1 \leq j \leq n$ e $\mathcal{A}'(A_{n+1}) = 0$ si ha che $\mathcal{A}' \models F$. Infatti, per ogni clausola C_i di F , o $C_i \in \overline{F}_0$ (e in questo caso $\mathcal{A}' \models C_i$ se e solo se $\mathcal{A}' \models C_i \setminus \{A_{n+1}\}$), oppure C_i contiene $\neg A_{n+1}$ e automaticamente $\mathcal{A}' \models C_i$. Dunque anche F sarebbe soddisfacibile: ma questo contraddice la nostra ipotesi! Analogamente si dimostra che se F_1 è soddisfacibile allora anche F lo è, nuovamente una contraddizione. Dunque sia F_0 che F_1 sono insoddisfacibili e, per l'ipotesi induttiva (siccome F_0 e F_1 utilizzano soltanto le lettere proposizionali A_1, \dots, A_n), si ha che $\emptyset \in Res^*(F_0)$ e $\emptyset \in Res^*(F_1)$. Poiché F_0 è stato definito a partire da \overline{F}_0 eliminando A_{n+1} da ogni clausola e si può derivare \emptyset da F_0 mediante risoluzione, allora si può derivare o \emptyset oppure $\{A_{n+1}\}$ da \overline{F}_0 (semplicemente reinserendo A_{n+1} ove necessario nella derivazione per risoluzione in questione). Analogamente, si può derivare o \emptyset oppure $\{\neg A_{n+1}\}$ da \overline{F}_1 . Se si può derivare \emptyset da qualche \overline{F}_i allora $\emptyset \in Res^*(F)$ poiché $F = \overline{F}_0 \cup \overline{F}_1$ e dunque $Res^*(\overline{F}_0) \subseteq Res^*(F)$. Altrimenti si può derivare $\{A_{n+1}\}$ da \overline{F}_0 e $\{\neg A_{n+1}\}$ da \overline{F}_1 : ma la risolvente di queste due clausole è proprio \emptyset e dunque $\emptyset \in Res^*(F)$ nuovamente. \square

Teorema 12 (Completezza della logica proposizionale (versione finita)).
Siano F e G due formule proposizionali qualsiasi. Sia H la forma normale congiuntiva di $F \wedge \neg G$ ($\equiv \neg(F \rightarrow G)$) ottenuta mediante l'algoritmo CNF. Le affermazioni seguenti sono equivalenti:

1. $F \models G$
2. $\{F\} \vdash G$
3. $\emptyset \in Res^*(H)$.

Dimostrazione. 2 implica 1 per il teorema di correttezza (per le derivazioni formali). Mentre 1 implica 3 per il precedente Teorema (completezza della risoluzione). Resta da provare che 3 implica 2. Si ricordi che $F \wedge \neg G \dashv\vdash H$. Inoltre, poiché $\emptyset \in Res^*(H)$ si deve anche avere $\{A\}, \{\neg A\} \in Res^*(H)$ per qualche formula (atomica) A . Ma per quanto visto nella dimostrazione del teorema di correttezza per la risoluzione, questo implica che $\{H\} \vdash A$ e $\{H\} \vdash \neg A$. Dunque

1. $\{F, \neg G\} \vdash F$ Ass
2. $\{F, \neg G\} \vdash \neg G$ Ass
3. $\{F, \neg G\} \vdash F \wedge \neg G$ \wedge -I su 1 e 2
4. $\{F, \neg G\} \vdash H$ poiché $F \wedge \neg G \dashv\vdash H$
5. $\{F, \neg G\} \vdash A$ poiché $\{H\} \vdash A$
6. $\{F, \neg G\} \vdash \neg A$ poiché $\{H\} \vdash \neg A$
7. $\{F\} \vdash \neg\neg G$ Assurdo su 5 e 6
8. $\{F\} \vdash G$ DN-E su 7 □

Il metodo della risoluzione costituisce un algoritmo per risolvere il problema della soddisfacibilità per la logica proposizionale: infatti, presa una qualunque formula F la si trasforma in CNF e poi si applica il metodo della risoluzione (dopo aver eventualmente eliminato tutte le clausole che sono tautologie). Si osservi che, in questo caso, $Res^*(F)$ è un insieme finito e quindi può essere determinato completamente in un numero finito di passi: fatto ciò basta controllare se $\emptyset \in Res^*(F)$ e concludere “ H è insoddisfacibile” se ciò avviene e “ H è soddisfacibile” altrimenti.

Si osservi, però, che anche l’algoritmo del metodo della risoluzione non lavora in tempo polinomiale.

Esercizi.

1. Verificare col metodo della risoluzione ciascuna delle seguenti affermazioni:
 - (a) $\neg A$ è una conseguenza di $(A \rightarrow B) \wedge (A \rightarrow \neg B)$
 - (b) $(\neg A \wedge \neg B \wedge C) \vee (\neg A \wedge \neg C) \vee (B \wedge C) \vee A$ è una tautologia
 - (c) $((A \rightarrow B) \wedge (A \rightarrow \neg B)) \rightarrow \neg A$ è una tautologia.
 - (d) $(A \vee B) \rightarrow \neg(A \rightarrow (B \rightarrow C))$ è soddisfacibile¹⁴
 - (e) $A \rightarrow C$ non è conseguenza logica di $A \wedge B \rightarrow C$.
2. Verificare col metodo della risoluzione se le seguenti formule sono tautologie, contraddizioni o nessuna delle due:
 - $(\neg A \rightarrow B) \vee ((A \wedge \neg C) \leftrightarrow B)$
 - $(A \rightarrow B) \wedge (A \rightarrow \neg B)$
 - $(A \rightarrow B \vee C) \vee (C \rightarrow \neg A)$
 - $((A \rightarrow B) \wedge C) \vee (A \wedge D)$

¹⁴Si ricordi che questo punto e il seguente si possono risolvere perché, nel caso della logica proposizionale, l’algoritmo per trovare $Res^*(F)$ termina *sempre* in un numero finito di passi. Vedremo che nel caso della logica predicativa del prim’ordine questo non è vero, e quindi la risoluzione potrà solo rispondere “No, F non è soddisfacibile”, ma mai “ F è soddisfacibile”.

3. Verificare col metodo della risoluzione se le seguenti formule proposizionali sono tautologie:

$$(A \vee B) \wedge (C \rightarrow \neg A) \rightarrow (C \rightarrow B)$$

$$((A \rightarrow \neg A) \wedge (B \rightarrow A)) \rightarrow \neg B$$

$$(A \wedge B) \vee (\neg A \wedge B) \vee B$$

$$(A \wedge B) \vee (\neg A \wedge B) \vee \neg B$$

4. Verificare col metodo della risoluzione se le seguenti formule sono insoddisfacibili:

$$((A \vee B) \wedge (\neg A \vee B) \wedge \neg B) \rightarrow B$$

$$(A \rightarrow \neg B) \wedge (\neg A \vee B)$$

$$(A \rightarrow \neg B) \wedge (\neg A \wedge B)$$

$$(A \rightarrow \neg B) \wedge A \wedge B$$

$$(A \vee \neg B \vee C) \wedge \neg C \wedge (\neg A \vee B) \wedge \neg A$$

Vogliamo ora dimostrare la completezza della logica proposizionale, ovvero che $\mathcal{F} \vdash G$ se e solo se $\mathcal{F} \models G$. Il fatto che da $\mathcal{F} \vdash G$ segue $\mathcal{F} \models G$ è esattamente il Teorema di Correttezza già dimostrato nelle sezioni precedenti. Per quanto riguarda l'altra direzione essa è stata già dimostrata nel caso in cui \mathcal{F} sia un insieme *finito* di formule: in questo caso, infatti, $\mathcal{F} = \{F_1, \dots, F_n\}$ e si ha che $\mathcal{F} \models G$ se e solo se $\bigwedge_{i=1}^n F_i \models G$ se e solo se (per la versione finita del Teorema di Completezza della logica proposizionale) $\{\bigwedge_{i=1}^n F_i\} \vdash G$ se e solo se $\mathcal{F} \vdash G$. Dunque bisogna capire come passare da insiemi di formule finiti ad arbitrari insiemi di formule. Questo passaggio è dato dal teorema di compattezza.

Lemma 13 (di König). *Sia X un insieme infinito di stringhe binarie finite (ovvero di sequenze finite di 0 e 1). Allora esiste una stringa binaria infinita \bar{w} tale che ogni segmento iniziale di \bar{w} è anche un segmento iniziale di una quantità infinita di $\bar{x} \in X$.*

Dimostrazione. Costruiremo induttivamente (cifra per cifra) la stringa \bar{w} ed una sequenza di insiemi *infiniti*

$$X = X_0 \supseteq X_1 \supseteq X_2 \supseteq \dots \supseteq X_n \supseteq \dots$$

tali che per ogni n se t è l'unico segmento iniziale di \bar{w} di lunghezza n allora t è un segmento iniziale di tutte le stringhe in X_n .

Per stabilire la prima cifra di \bar{w} consideriamo la prima cifra di tutte le sequenze in $X = X_0$. Poiché esse sono infinite e possono essere solamente o

0 o 1 vi sono due possibilità: o vi sono infinite stringhe che hanno 0 come prima cifra oppure no (e in questo secondo caso ve ne saranno infinite che cominciano per 1). Nel primo caso stabiliamo che la prima cifra di \bar{w} sia 0 e definiamo

$$X_1 = \{\bar{x} \in X_0 \mid \text{la prima cifra di } \bar{x} \text{ è uno } 0\},$$

mentre nel secondo caso stabiliamo che la prima cifra di \bar{w} sia 1 e definiamo

$$X_1 = \{\bar{x} \in X_0 \mid \text{la prima cifra di } \bar{x} \text{ è un } 1\}.$$

Si osservi che in ogni caso x_1 è un sottoinsieme infinito di X e che (comunque vengano definite le altre cifre di \bar{w}), il segmento iniziale di \bar{w} di lunghezza 1 sarà sempre segmento iniziale di infinite sequenze \bar{x} di X (e precisamente di quelle \bar{x} che stanno in X_1).

Supponiamo ora di aver definito le prime n cifre di \bar{w} e di aver definito $X_n \subseteq X$ in maniera tale che esso sia infinito e che il segmento iniziale di \bar{w} di lunghezza n sia anche segmento iniziale di tutte le sequenze in X_n . Per stabilire la $n + 1$ -esima cifra di \bar{w} consideriamo la $n + 1$ -esima cifra di tutte le sequenze in X_n (che abbiano lunghezza maggiore di n). Poiché vi sono infinite sequenze di questo genere, abbiamo nuovamente due possibilità: se ve ne sono infinite che hanno 0 come $n + 1$ -esima cifra stabiliamo che 0 sia anche la $n + 1$ -esima cifra di \bar{w} e poniamo

$$X_{n+1} = \{\bar{x} \in X_n \mid \text{la } n + 1\text{-esima cifra di } \bar{x} \text{ è uno } 0\},$$

altrimenti ve ne devono essere infinite che hanno 1 come $n + 1$ -esima cifra e allora stabiliamo che 1 sia anche la $n + 1$ -esima cifra di \bar{w} e poniamo

$$X_{n+1} = \{\bar{x} \in X_n \mid \text{la } n + 1\text{-esima cifra di } \bar{x} \text{ è un } 1\}.$$

In ogni caso si può di nuovo verificare che $X_{n+1} \subseteq X_n \subseteq X$ è un insieme infinito e che il segmento iniziale di \bar{w} di lunghezza $n + 1$ è anche segmento iniziale di infinite $\bar{x} \in X$ (ovvero di quelle che stanno in X_{n+1}). \square

Teorema 14 (Compattezza della logica proposizionale). *Sia $\mathcal{F} = \{F_1, F_2, \dots\}$ un insieme infinito di formule proposizionali. Allora \mathcal{F} è insoddisfacibile se e solo se qualche sottoinsieme finito di \mathcal{F} è insoddisfacibile.*

Equivalentemente se \mathcal{F} è finitamente soddisfacibile (cioè ogni suo sottoinsieme finito è soddisfacibile) allora è soddisfacibile.

Dimostrazione. Dimostreremo la seconda parte dell'enunciato (a cui la prima è equivalente). Una direzione è ovvia: se esiste un assegnamento \mathcal{A} che rende vere tutte le formule di \mathcal{F} , allora, in particolare, \mathcal{A} rende vere tutte le formule di un qualunque $\mathcal{F}' \subseteq \mathcal{F}$.

Viceversa, sia A_1, A_2, \dots una enumerazione (eventualmente infinita) di tutte le lettere proposizionali che occorrono in qualche $F_i \in \mathcal{F}$. Se ogni

sottoinsieme finito di \mathcal{F} è soddisfacibile, allora per ogni $n > 0$ esiste un assegnamento \mathcal{A}_n definito su A_1, \dots, A_{k_n} tale che $\mathcal{A}_n \models \bigwedge_{i=1}^n F_i$ (dove k_n è un numero tale che $\mathcal{L}(F_i) \subseteq \{A_1, \dots, A_{k_n}\}$ per ogni $1 \leq i \leq n$). In particolare, ogni formula F_i di \mathcal{F} è modellata da tutti gli assegnamenti \mathcal{A}_n tali che $n \geq i$. Si osservi inoltre che i valori che ciascun \mathcal{A}_n assegna alle lettere proposizionali A_1, \dots, A_{k_n} formano una stringa binaria finita x_n . Sia $X = \{x_n \mid n > 0\}$. Se X è finito allora esiste un assegnamento \mathcal{A} tale che $\mathcal{A}_n = \mathcal{A}$ per infiniti indici n . Ma allora $\mathcal{A} \models F_i$ per ogni $i \geq 1$, poiché deve esistere un $n \geq i$ tale che $\mathcal{A} = \mathcal{A}_n \models F_i$. Se invece X è infinito possiamo applicare il Lemma di König per trovare una stringa binaria infinita \bar{w} tale che ogni suo segmento iniziale è anche segmento iniziale di infinite x_n . Definiamo ora l'assegnamento \mathcal{A} ponendo $\mathcal{A}(A_j) = 0$ se la j -esima cifra di \bar{w} è 0, e $\mathcal{A}(A_j) = 1$ altrimenti. Sia ora F_i una formula di \mathcal{F} e sia m tale che $\mathcal{L}(F_i) \subseteq \{A_1, \dots, A_m\}$. Sia $n \geq i$ tale che il segmento iniziale t di lunghezza m di \bar{w} sia anche segmento iniziale di x_n (tale n esiste perché vi sono *infiniti* indici k per cui t è segmento iniziale di x_k). Poiché $\mathcal{A}(A_j) = \mathcal{A}_n(A_j)$ per ogni $1 \leq j \leq m$ ed il valore che un assegnamento dà a F_i dipende solo da tali valori si ha che $\mathcal{A} \models F_i$ se e solo se $\mathcal{A}_n \models F_i$. Ma poiché $n \geq i$ si deve avere $\mathcal{A}_n \models F_i$. Dunque \mathcal{A} rende vera ogni formula di \mathcal{F} e \mathcal{F} è soddisfacibile. \square

Osservazione. La compattezza è una proprietà importante che non vale per qualunque linguaggio. Si considerino per esempio le seguenti frasi:

$F_0 =$ “Ci sono una quantità finita di oggetti nell’universo”

$F_1 =$ “C’è almeno un oggetto nell’universo”

$F_2 =$ “Ci sono almeno due oggetti nell’universo”

\vdots

$F_n =$ “Ci sono almeno n oggetti nell’universo”

\vdots

Prese tutte insieme queste frasi sono contraddittorie (se F_0 è vera allora deve esistere un n tale che ci sono esattamente n oggetti nell’universo: ma allora F_{n+1} è falsa!). Tuttavia, se noi prendiamo solo un numero *finito* di queste frasi esse non sono più contraddittorie. Ogni insieme finito di queste frasi è soddisfacibile, ma l’intera collezione no. Dunque qualunque linguaggio (e qualunque logica) che sia in grado di esprimere queste frasi non ha la proprietà di compattezza.

Consideriamo ora un insieme di formule \mathcal{F} (finito o infinito). Le formule di \mathcal{F} si possono supporre tutte in CNF, e quindi \mathcal{F} può essere visto complessivamente come un insieme di clausole.¹⁵ Tuttavia, se partiamo da una quantità infinita di formule, tale insieme sarà infinito e non potrà essere visto come una sola formula in CNF.

¹⁵Chiaramente, se \mathcal{F} è finito allora esiste una singola formula G (la congiunzione $\bigwedge \mathcal{F}$ delle formule in \mathcal{F}) tale che $\mathcal{F} = G$ (visti come insiemi di clausole).

Nonostante ciò, si possono definire gli insiemi $Res^n(\mathcal{F})$ per ogni $n \in \mathbb{N}$, e porre poi $Res^*(\mathcal{F}) = \bigcup_{n=0}^{\infty} Res^n(\mathcal{F})$. Grazie al Teorema di Compattezza si può provare la seguente:

Proposizione 15. *Sia \mathcal{F} un insieme (finito o infinito) di formule in CNF. Allora \mathcal{F} è insoddisfacibile se e solo se $\emptyset \in Res^*(\mathcal{F})$.*

Dimostrazione. Per il Teorema di Compattezza si ha che \mathcal{F} è insoddisfacibile se e solo se \mathcal{F}_0 è insoddisfacibile per qualche $\mathcal{F}_0 \subseteq \mathcal{F}$ finito. Inoltre \mathcal{F}_0 è insoddisfacibile se e solo se $\bigwedge \mathcal{F}_0$ lo è e per il teorema di completezza della risoluzione questo è vero se e solo se $\emptyset \in Res^*(\bigwedge \mathcal{F}_0) = Res^*(\mathcal{F}_0) \subseteq Res^*(\mathcal{F})$.

Viceversa, se $\emptyset \in Res^*(\mathcal{F})$ allora deve esistere un n tale che $\emptyset \in Res^n(\mathcal{F})$. Per induzione su n si può dimostrare che per ogni elemento di $Res^*(\mathcal{F})$ esiste un numero *finito* di clausole di \mathcal{F} da cui esso è stato derivato. In particolare, esiste un numero finito di clausole di \mathcal{F} da cui \emptyset è stato derivato e quindi esiste un insieme finito di formule $\mathcal{F}_0 \subseteq \mathcal{F}$ tale che $\emptyset \in Res^*(\mathcal{F}_0)$. Ma, di nuovo per la completezza della risoluzione e la compattezza della logica proposizionale, questo significa che \mathcal{F} è insoddisfacibile. \square

Da questo segue il

Teorema 16 (Completezza della logica proposizionale). *Per ogni formula G e per ogni insieme di formule \mathcal{F} , $\mathcal{F} \models G$ se e solo se $\mathcal{F} \vdash G$.*

Dimostrazione. Per il teorema di correttezza se $\mathcal{F} \vdash G$ allora $\mathcal{F} \models G$.

Viceversa, se $\mathcal{F} \models G$ allora $\mathcal{F} \cup \{\neg G\}$ è insoddisfacibile. Per il teorema di compattezza deve esistere un insieme *finito* $\mathcal{F}_0 \subseteq \mathcal{F}$ tale che $\mathcal{F}_0 \cup \{\neg G\}$ è insoddisfacibile (equivalentemente, $\mathcal{F}_0 \models G$). Poiché ciò è equivalente a $\bigwedge \mathcal{F}_0 \models G$, per la versione finita del teorema di completezza della logica proposizionale si ha anche $\{\bigwedge \mathcal{F}_0\} \vdash G$ e dunque $\mathcal{F}_0 \vdash G$. Ma per la regola di monotonia, questo implica anche che $\mathcal{F} \vdash G$. \square

Esercizi.

1. Supponiamo di rimuovere \vee -eliminazione, \vee -simmetria, \rightarrow -introduzione e \rightarrow -eliminazione dalla lista delle regole di derivazione formale, e di rimpiazzarle con Leggi di De Morgan, \vee -distributività, Regola di Taglio, Regola dell'Assurdo e Eliminazione della Doppia Negazione. Dimostrare che l'insieme di regole di derivazione risultante è completo.
2. Siano \mathcal{F} e \mathcal{G} insiemi di formule. Diciamo che \mathcal{F} è (logicamente) equivalente a \mathcal{G} (e scriviamo $\mathcal{F} \equiv \mathcal{G}$) se e solo se per ogni assegnamento \mathcal{A} , $\mathcal{A} \models \mathcal{F}$ se e solo se $\mathcal{A} \models \mathcal{G}$.
 - (a) Mostrare che per ogni \mathcal{F} e \mathcal{G} , $\mathcal{F} \equiv \mathcal{G}$ se e solo se sia $\mathcal{F} \vdash G$ per ogni $G \in \mathcal{G}$, sia $\mathcal{G} \vdash F$ per ogni $F \in \mathcal{F}$.

(b) Dimostrare che non è vero che per ogni \mathcal{F} e \mathcal{G} , $\mathcal{F} \equiv \mathcal{G}$ se e solo se valgono entrambe le seguenti:

- per ogni $G \in \mathcal{G}$ esiste $F \in \mathcal{F}$ tale che $G \models F$;
- per ogni $F \in \mathcal{F}$ esiste $G \in \mathcal{G}$ tale che $F \models G$.

3. Se da un insieme di formule \mathcal{F} si può derivare una contraddizione, allora \mathcal{F} si dice *inconsistente*. Nel caso contrario, \mathcal{F} viene detto *consistente*.

Dimostrare che \mathcal{F} è consistente se e solo se è soddisfacibile.

4. Supponiamo che \mathcal{F} sia un insieme di formule inconsistente. Per ogni $G \in \mathcal{F}$, sia \mathcal{F}_G l'insieme $\mathcal{F} \setminus \{G\}$.

(a) Dimostrare che per ogni $G \in \mathcal{F}$, $\mathcal{F}_G \vdash \neg G$ usando l'esercizio precedente.

(b) Dimostrare che per ogni $G \in \mathcal{F}$, $\mathcal{F}_G \vdash \neg G$ tracciando una dimostrazione formale (ovvero: *fissato* un certo G derivare formalmente G da \mathcal{F}_G).

5. Un insieme di formule \mathcal{F} si dice *chiuso per congiunzioni* se \mathcal{F} è chiuso per $f \wedge$, ovvero se per ogni F e G in \mathcal{F} , $F \wedge G \in \mathcal{F}$.

Supponiamo che \mathcal{F} sia chiuso per congiunzioni e inconsistente. Dimostrare che per ogni $G \in \mathcal{F}$ esiste $F \in \mathcal{F}$ tale che $\{F\} \vdash \neg G$.

6. Un insieme di formule si dice *insoddisfacibile minimale* se è insoddisfacibile, ma ogni suo sottoinsieme proprio è soddisfacibile.

(a) Dimostrare che per ogni $n \in \mathbb{N}$ esistono insiemi di formule insoddisfacibili minimali contenenti esattamente n formule.

(b) Dimostrare che ogni insieme di formule insoddisfacibile contiene un sottoinsieme insoddisfacibile minimale.