

# Modelli della Concorrenza

UniShare

Davide Cozzi  
@dlcgold

# Indice

<b>1</b>	<b>Introduzione</b>	<b>2</b>
<b>2</b>	<b>Introduzione alla concorrenza</b>	<b>3</b>
<b>3</b>	<b>Ripasso</b>	<b>5</b>
3.1	Ripasso logica proposizionale . . . . .	5
3.1.1	Sintassi . . . . .	6
3.1.2	Semantica . . . . .	8
3.1.3	Equivalenze Logiche . . . . .	9
<b>4</b>	<b>Correttezza di programmi sequenziali</b>	<b>11</b>

# Capitolo 1

## Introduzione

Questi appunti sono presi a lezione. Per quanto sia stata fatta una revisione è altamente probabile (praticamente certo) che possano contenere errori, sia di stampa che di vero e proprio contenuto. Per eventuali proposte di correzione effettuare una pull request. Link: <https://github.com/dlcgold/Appunti>.

Le immagini presenti in questi appunti sono tratte dalle slides del corso e tutti i diritti delle stesse sono da destinarsi ai docenti del corso stesso.

## Capitolo 2

# Introduzione alla concorrenza

La **concorrenza** è presente in diversi aspetti della quotidianità. Un primo esempio di **sistema concorrente** non legato all'informatica è la *cellula vivente*: può essere vista come un dispositivo che trasforma e manipola dati per ottenere un risultato. I vari processi all'interno di una cellula avvengono in modo concorrente, il che la rende un *sistema asincrono*. Un secondo esempio può essere quello dell'*orchestra musicale*: i vari componenti suonano spesso simultaneamente rappresentando un *sistema sincrono* (ovvero un sistema che funziona avendo una sorta di “cronometro” condiviso dai vari attori). Un esempio informatico invece è un *processore multicore* (anche se in realtà anche se fosse *monocore* sarebbe comunque un sistema concorrente per ovvie ragioni). Anche una *rete di calcolatori* è un modello concorrente, nonché i *modelli sociali umani*.

I modelli concorrenti hanno alcuni aspetti comuni, tra cui:

- competizione per l'accesso alle risorse condivise
- cooperazione per un fine comune (che può portare a competizione)
- coordinamento di attività diverse
- sincronia e asincronia

Durante lo studio e la progettazione di sistemi concorrenti si hanno diversi problemi peculiari che rendono il tutto molto complesso. Un sistema concorrente mal progettato può avere effetti catastrofici.

Per poter sviluppare modelli concorrenti si necessita innanzitutto di:

- **linguaggi**, per specificare e rappresentare sistemi concorrenti.
  - **linguaggi di programmazione** (con l'uso di *thread*, *mutex*, scambio di messaggi, etc. . . con i vari problemi di *race condition*, uso di variabili condivise etc. . .).

- linguaggi rappresentativi, come ad esempio il caso di una *partitura musicale* (dove si visualizza bene la natura *sincrona*).
  - **task graph** (*grafo delle attività*), dove i nodi sono le attività (o eventi) mentre gli archi rappresentano una *relazione d'ordine parziale*, come per esempio una *relazione di precedenza*, sui nodi.
  - **algebre di processi**, simile ad un sistema di equazioni, con simboli che rappresentano eventi del sistema concorrente e operatori atti a comporre fra loro i vari sottoprocessi del sistema concorrente. Ogni “equazione” descrive un processo che costituisce un elemento di un sistema concorrente.
  - **modelli**, per modellare sistemi concorrenti in astratto. Un esempio è dato dalle **reti di Petri**, che modellano un sistema concorrente partendo dalle nozioni di *stato locale* di uno dei componenti del sistema e di *evento locale* che ha un effetto su alcune componenti (e non tutte). Si ha quindi rappresentato un *sistema dinamico* che si evolve nel tempo (evoluzione rappresentata tramite *relazioni di flusso*).
- **logica**, per analizzare e specificare sistemi concorrenti.
  - **model-checking**, per validare formule relative a proprietà di sistemi concorrenti.

# Capitolo 3

## Ripasso

### 3.1 Ripasso logica proposizionale

Si ringrazia [Marco Natali](#)<sup>1</sup> per questo ripasso.

La logica è lo studio del ragionamento e dell'argomentazione e, in particolare, dei procedimenti inferenziali, rivolti a chiarire quali procedimenti di pensiero siano validi e quali no. Vi sono molteplici tipologie di logiche, come ad esempio la logica classica e le logiche costruttive, tutte accomunate dall'essere composte da 3 elementi:

- **Linguaggio:** insieme di simboli utilizzati nella Logica per definire le cose.
- **Sintassi:** insieme di regole che determina quali elementi appartengono o meno al linguaggio.
- **Semantica:** permette di dare un significato alle formule del linguaggio e determinare se rappresentano o meno la verità.

Ci occupiamo della logica Classica che si compone in *logica proposizionale* e *logica predicativa*.

La logica proposizionale è un tipo di logica classica che presenta come caratteristica quella di essere un linguaggio limitato, ovvero caratterizzato dal poter esprimere soltanto proposizioni senza possibilità di estensione ad una classe di persone.

---

<sup>1</sup>Link al repository: <https://github.com/bigboss98/Appunti-1/tree/master/PrimoAnno/Fondamenti>

### 3.1.1 Sintassi

Il linguaggio di una logica proposizionale è composto dai seguenti elementi:

- Variabili Proposizionali atomiche (o elementari):  $P, Q, R, p_i, \dots$
- Connettivi Proposizionali:  $\wedge, \vee, \neg, \implies, \iff$
- Simboli Ausiliari: “(“ e “)” (detti delimitatori)
- Costanti:  $T$  (*True*, *Vero*,  $\top$ ) e  $F$  (*False*, *Falso*,  $\perp$ )

La sintassi di un linguaggio è composta da una serie di formule ben formate ( $FBF$ ) definite induttivamente nel seguente modo:

1. Le costanti e le variabili proposizionali:  $\top, \perp, p_i \in FBF$ .
2. Se  $A$  e  $B \in FBF$  allora  $(A \wedge B), (A \vee B), (\neg A), (A \implies B), (A \iff B)$  sono delle formule ben formate.
3. nient'altro è una formula

**In una formula ben formata le parentesi sono bilanciate.**

**Esempio 1.** *Vediamo degli esempi:*

- $(P \wedge Q) \in FBF$  è una formula ben formata
- $(PQ \wedge R) \notin FBF$  in quanto non si rispetta la sintassi del linguaggio definita.

**Definizione 1.** *Sia  $A \in FBF$ , l'insieme delle sottoformule di  $A$  è definito come segue:*

1. *Se  $A$  è una costante o variabile proposizionale allora  $A$  stessa è la sua sottoformula.*
2. *Se  $A$  è una formula del tipo  $(\neg A')$  allora le sottoformule di  $A$  sono  $A$  stessa e le sottoformule di  $A'$ ;  $\neg$  è detto connettivo principale e  $A'$  sottoformula immediata di  $A$ .*
3. *Se  $A$  è una formula del tipo  $B \circ C$ , allora le sottoformule di  $A$  sono  $A$  stessa e le sottoformule di  $B$  e  $C$ ;  $\circ$  è il connettivo principale e  $B$  e  $C$  sono le due sottoformule immediate di  $A$ .*

È possibile ridurre ed eliminare delle parentesi attraverso l'introduzione della precedenza tra gli operatori, definita come segue:

$$\neg, \wedge, \vee, \implies, \iff$$

In assenza di parentesi una formula va parentizzata privilegiando le sottoformule i cui connettivi principali hanno la precedenza più alta.

In caso di parità di precedenza vi è la convenzione di associare da destra a sinistra.

Esempio:

$\neg A \wedge (\neg B \implies C) \vee D$  diventa  $((\neg A) \wedge ((\neg B) \implies C) \vee D)$ .

**Definizione 2.** *Un albero sintattico  $T$  è un albero binario coi nodi etichettati da simboli di  $L$ , che rappresenta la scomposizione di una formula ben formata  $X$  definita come segue:*

1. Se  $X$  è una formula atomica, l'albero binario che la rappresenta è composto soltanto dal nodo etichettato con  $X$
2. Se  $X = A \circ B$ ,  $X$  è rappresentata da un albero binario che ha la radice etichettata con  $\circ$ , i cui figli sinistri e destri sono la rappresentazione di  $A$  e  $B$
3. Se  $X = \neg A$ ,  $X$  è rappresentato dall'albero binario con radice etichettata con  $\neg$ , il cui figlio è la rappresentazione di  $A$

Poiché una formula è definita mediante un albero sintattico, le proprietà di una formula possono essere dimostrate mediante induzione strutturale sulla formula, ossia dimostrare che la proprietà di una formula soddisfi i seguenti 3 casi:

- è verificata la proprietà per tutte le formule atomo  $A$
- supposta verifica la proprietà per  $A$ , si verifica che la proprietà è verificata per  $\neg A$
- supposta la proprietà verificata per  $A_1$  e  $A_2$ , si verifica che la proprietà è verifica per  $A_1 \circ A_2$ , per ogni connettivo  $\circ$ .



### 3.1.2 Semantica

La semantica di una logica consente di dare un significato e un'interpretazione alle formule del Linguaggio.

**Definizione 3.** Sia data una formula proposizionale  $P$  e sia  $P_1, \dots, P_n$ , l'insieme degli atomi che compaiono nella formula  $A$ . Si definisce come interpretazione una funzione  $v : \{P_1, \dots, P_n\} \mapsto \{T, F\}$  che attribuisce un valore di verità a ciascun atomo della formula  $A$ .

$v : P \rightarrow \{0, 1\}$  è un'**assegnazione booleana**

I connettivi della Logica Proposizionale hanno i seguenti valori di verità:

A	B	$A \wedge B$	$A \vee B$	$\neg A$	$A \implies B$	$A \iff B$
$F$	$F$	$F$	$F$	$T$	$T$	$T$
$F$	$T$	$F$	$T$	$T$	$T$	$F$
$T$	$F$	$F$	$T$	$F$	$F$	$F$
$T$	$T$	$T$	$T$	$F$	$T$	$T$

Essendo ogni formula  $A$  definita mediante un unico albero sintattico, l'interpretazione  $v$  è ben definita e ciò comporta che data una formula  $A$  e un'interpretazione  $v$ , eseguendo la definizione induttiva dei valori di verità, si ottiene un unico  $v(A)$ .

**Definizione 4.** Una formula nella logica proposizionale può essere di diversi tipi:

- **Valida o Tautologica:** la formula è soddisfatta da qualsiasi valutazione della Formula
- **Soddisfacibile NON Tautologica:** la formula è soddisfatta da qualche valutazione della formula ma non da tutte.
- **Falsificabile:** la formula non è soddisfatta da qualche valutazione della formula.
- **Contraddizione:** la formula non viene mai soddisfatta

**Teorema 1.** Si ha che:

- $A$  è una formula valida se e solo se  $\neg A$  è insoddisfacibile.
- $A$  è soddisfacibile se e solo se  $\neg A$  è falsificabile

### Modelli e decidibilità

Si definisce *modello*, indicato con  $M \models A$ , tutte le valutazioni booleane che rendono vera la formula  $A$ . Si definisce *contromodello*, indicato con  $M \not\models A$ , tutte le valutazioni booleane che rendono falsa la formula  $A$ .

La logica proposizionale è decidibile (posso sempre verificare il significato di una formula). Esiste infatti una procedura effettiva che stabilisce la validità o no di una formula, o se questa ad esempio è una tautologia. In particolare il verificare se una proposizione è tautologica o meno è l'operazione di decidibilità principale che si svolge nel calcolo proposizionale.

**Definizione 5.** Se  $M \models A$  per tutti gli  $M$ , allora  $A$  è una tautologia e si indica  $\models A$

**Definizione 6.** Se  $M \models A$  per qualche  $M$ , allora  $A$  è soddisfacibile

**Definizione 7.** Se  $M \not\models A$  non è soddisfatta da nessun  $M$ , allora  $A$  è insoddisfacibile

### 3.1.3 Equivalenze Logiche

Date due formule  $A$  e  $B$ , si dice che  $A$  è *logicamente equivalente* a  $B$ , indicato con  $A \equiv B$ , se e solo se per ogni interpretazione  $v$  risulta  $v(A) = v(B)$ .

Nella logica proposizionale sono definite le seguenti equivalenze logiche, indicate con  $\equiv$ :

#### 1. Idempotenza:

$$A \vee A \equiv A$$

$$A \wedge A \equiv A$$

#### 2. Associatività:

$$A \vee (B \vee C) \equiv (A \vee B) \vee C$$

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

#### 3. Commutatività:

$$A \vee B \equiv B \vee A$$

$$A \wedge B \equiv B \wedge A$$

**4. Distributività:**

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

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

**5. Assorbimento:**

$$A \vee (A \wedge B) \equiv A \quad A \wedge (A \vee B) \equiv A$$

**6. Doppia negazione:**

$$\neg\neg A \equiv A$$

**7. Leggi di De Morgan:**

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

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

**8. Terzo escluso:**

$$A \vee \neg A \equiv T$$

**9. Contrapposizione:**

$$A \implies B \equiv \neg B \implies \neg A$$

**10. Contraddizione**

$$A \wedge \neg A \equiv F$$

**Completezza di insiemi di Connettivi**

Un insieme di connettivi logici è completo se mediante i suoi connettivi si può esprimere un qualunque altro connettivo. Nella logica proposizionale valgono anche le seguenti equivalenze, utili per ridurre il linguaggio:

$$(A \implies B) \equiv (\neg A \vee B)$$

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

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

$$(A \iff B) \equiv (A \implies B) \wedge (B \implies A)$$

L'insieme dei connettivi  $\{\neg, \vee, \wedge\}$ ,  $\{\neg, \wedge\}$  e  $\{\neg, \vee\}$  sono completi.

## Capitolo 4

# Correttezza di programmi sequenziali

Introduciamo l'argomento con un esempio.

**Esempio 2.** *Definiamo una funzione in C che riceve un intero (la lunghezza del vettore) e un vettore e restituisce un numero intero. Iniziamo a chiederci*

---

**Listing 1** Esempio di funzione in C

---

```
int f(int n, const int v[]) {
    int x = v[0];
    int h = 1;
    while (h < n) {
        if (x < v[h])
            x = v[h];
        h = h + 1;
    }
    return x;
}
```

---

*cosa fa la funzione e scopriamo che cerca il massimo in un vettore. Si può cercare una dimostrazione del codice.*

*Ricostruiamo la strategia della funzione, ovvero spostarsi lungo il vettore e conservare in  $x$  il valore massimo fino ad ora trovato. Arrivati alla fine del vettore so che in  $x$  avrò il valore massimo.*

*Più formalmente suppongo che  $n$  sia  $n > 0$ , per dire che ho almeno un elemento nel vettore. Suppongo inoltre che  $v[i] \in \mathbb{Z}$ ,  $\forall i \in \{0, \dots, n-1\}$ . Abbiamo fissato le **condizioni iniziali**.*

*All'inizio  $x$  è il massimo del sotto-vettore con solo il primo elemento ( $v[0..0]$ )*

e dopo l'assegnamento di  $h$  in  $v[0..h-1]$ , che, con  $h=1$  mi conferma che  $x$  è il massimo in  $v[0..0]$ .

Al termine di una certa iterazione  $x$  contiene il massimo contiene il massimo tra i valori compresi tra  $v[0]$  e  $v[h-1]$  (detto altrimenti il massimo in  $v[0..h-1]$ ). Inoltre al fine di una certa iterazione mi aspetto che  $h \leq n$ . Possiamo quindi dire che quando esco dal ciclo  $x$  è il massimo in  $v[0..h-1]$  ma in questo momento  $h=n$  e quindi  $x$  è il massimo del vettore.