Logica

4: Sintassi

Claudio Sacerdoti Coen

<sacerdot@cs.unibo.it>

Universitá di Bologna

25/10/2017

Outline

Sintassi

Wikipedia: “*La sintassi è la branca della linguistica che studia i diversi modi in cui le parole si uniscono tra loro per formare una proposizione ed i vari modi in cui le proposizioni si collegano per formare un periodo.*

Sintassi = descrizione dell’insieme di tutte le connotazioni alle quali associamo una denotazione (semantica)

Esempi:
- sintassi di un linguaggio di programmazione
- sintassi di una logica
- sintassi della lingua italiana

Piano di lavoro

Nelle scorse lezioni abbiamo capito che:

1. Il linguaggio naturale è ambiguo e affetto da paradossi
2. Vogliamo sviluppare un linguaggio artificiale per studiare i ragionamenti corretti
3. Dobbiamo definire la sintassi del linguaggio (le connotazioni)
4. Dobbiamo definire la semantica del linguaggio in funzione di un mondo (le denotazioni associate in un mondo alle connotazioni)
5. Dobbiamo verificare che valga il principio di invarianza per sostituzione
6. Dobbiamo studiare la conseguenza logica usando tale linguaggio

In questa lezione ci occupiamo del terzo punto (la sintassi)

Alfabeti, stringhe, linguaggi, grammatiche

Un alfabeto è un qualunque insieme non vuoto di simboli.
Esempi: $\{a, b, c, \ldots\}$, $\{0, 1\}$

Una stringa su un alfabeto è una qualunque sequenza finita, anche vuota, di simboli dell'alfabeto.
Esempi su $\{0, 1\}$: ϵ (stringa vuota), 0, 1, 00, 01, 10, 0110110

Un linguaggio su un alfabeto è un insieme di stringhe su quell'alfabeto.
Esempi su $\{0, 1\}$: $\{\epsilon, 0, 00, 000, \ldots\}$, $\{0, 1\}$

Una grammatica è un qualunque formalismo che definisce un linguaggio.

Backus-Naur Form (BNF)

La Backus-Naur Form (BNF) è una notazione per descrivere grammatiche.

Nota:

- non tutti i linguaggi sono descrivibili tramite BNF
- la sintassi dei linguaggi di programmazione è tipicamente data tramite BNF

Backus-Naur Form (BNF)

Una BNF è una quadrupla (T, NT, X, P) dove

1. T è un alfabeto ovvero un insieme non vuoto di simboli detti terminali
2. NT è un insieme non vuoto di simboli detti non terminali distinti da quelli di T
3. $X \in NT$, è il simbolo non terminale iniziale
4. P è un insieme di coppie (chiamate produzioni) formate da un non terminale e da un insieme di stringhe di simboli terminali o non terminali.
   La produzione $(X, \{\omega_1, \ldots, \omega_n\})$ si rappresenta come
   $X ::= \omega_1 \mid \ldots \mid \omega_n$ (che si legge X è ω_1 oppure $\ldots \omega_n$)

Esempio: $(\{0, 1\}, \{X, Y\}, X, \{X ::= 0 \mid 0Y,\ Y ::= 1X\})$

Backus-Naur Form (BNF)

Di solito di una BNF si indicano solamente le produzioni P:

- i simboli non terminali sono allora tutti quelli con cui iniziano le produzioni
- i simboli non terminali sono tutti i simboli delle produzioni esclusi i non terminali
- il simbolo iniziale è il simbolo con cui inizia la prima produzione

Esempio di prima $(\{0,1\}, \{X, Y\}, X, \{X ::= 0 \mid 0Y,\ \ Y ::= 1X\})$:

$X ::= 0 \mid 0Y$
$Y ::= 1X$

Non terminali: $\{X, Y\}$; Simbolo iniziale: X; Terminali: $\{0, 1\}$.

Backus-Naur Form (BNF)

Il linguaggio riconosciuto da una BNF (T, NT, X, P) si definisce nel modo seguente:

La stringa ω di soli terminali appartiene al linguaggio sse ottengo ω a partire da X rimpiazzando ripetutamente ciascun non terminale con una delle stringhe alternative a lui associate in una produzione.

Esempio di prima:

$X ::= 0 \mid 0Y$

$Y ::= 1X$

01010 appartiene al linguaggio poichè

$X \rightarrow 0Y \rightarrow 01X \rightarrow 010Y \rightarrow 0101X \rightarrow 01010$

000 non appartiene al linguaggio

Backus-Naur Form (BNF)

Esempio di prima:
$X ::= 0 \mid 0Y$
$Y ::= 1X$

Il linguaggio generato dalla BNF è $\{0, 010, 01010, 0101010, \ldots\}$

Altro esempio:
$X ::= aXa \mid bXb \mid \epsilon$

Il linguaggio generato dalla BNF è quello di tutte le stringhe palindrome di sole a e b:
$\{\epsilon, aa, bb, aaaa, abba, baab, bbbb, aaaaaa, \ldots\}$

Ambiguità

Una grammatica definita da una BNF è ambigua se si mostra in due modi diversi che una parola ω appartiene al linguaggio.

Esempio:

$F ::= x \mid y \mid F + F \mid F * F$

$x + y * x$ la posso ottenere in due modi diversi:

$F \mapsto F + F \mapsto x + F * F \mapsto x + y * x$

(che corrisponde a $x + (y * x)$)

$F \mapsto F * F \mapsto F + F * x \mapsto x + y * x$

(che corrisponde a $(x + y) * x$)

Noi siamo interessati solamente a grammatiche non ambigue.

Precedenza, associatività, parentesi

Per rendere le nostre grammatiche non ambigue, useremo le seguenti estensioni alle BNF:

1. Fissiamo un'ordine di precedenza fra operatori distinti
   Esempio: $* > +$ che significa che $x + y * x$ si legge $x + (y * x)$ e non $(x + y) * x$
2. Fissiamo un'associatività per ogni operatore
   Esempio: se $*$ associa a sinistra e $+$ a destra, allora $x * y * z + a + b$ significa $((x * y) * z) + (a + b)$
3. Introduciamo l'uso delle parentesi
   Esempio: $(x + y) * x$ è diverso da $x + y * x$

Nota: costruendo con attenzione le grammatiche non c'è bisogno di questi trucchetti che però semplificano il lavoro.

Struttura ricorsiva

Consideriamo una BNF (estesa) non ambigua.

Una stringa appartiene al linguaggio della BNF solo se è generata in un modo solo.

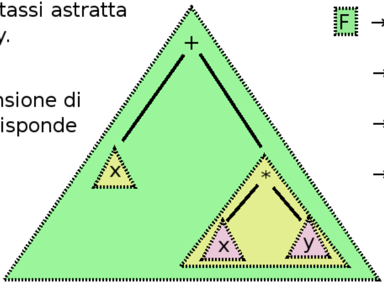
Ovvero a ogni stringa del linguaggio resta naturalmente associata una struttura ricorsiva (chiamato albero di sintassi astratta (AST)).

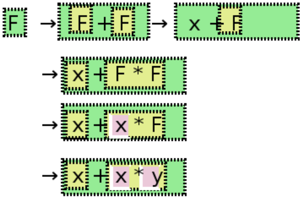
L'AST ha un nodo per ogni espansione di simbolo. La radice corrisponde al simbolo iniziale e le foglie alle espansioni fatte con solo simboli terminali. I nodi figli di un nodo corrispondono ai non terminali contenuti nell'espansione del nodo padre.

Albero di sintassi astratta

Albero di sintassi astratta per x + x * y.

A ogni espansione di simbolo corrisponde un nodo.



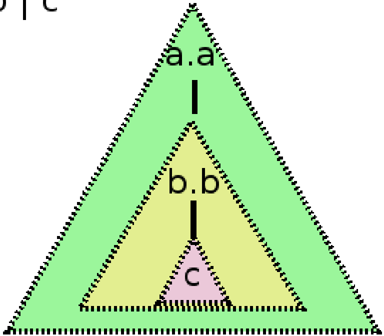


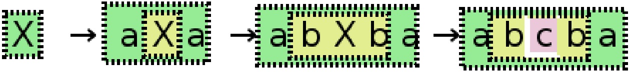
Le formule generate da un sottoalbero sono sottoformule immediate della formula generata dal nodo padre.

Esempio: x e $y * x$ sono sottoformule immediate solo di $x + y * x$ mentre y è sottoformula immediata solo di $x * y$.

Albero di sintassi astratta

X ::= aXa | bXb | c





Esempio: *bcb* è sottoformula immediata solo di *abcba* e *c* è sottoformula immediata di *bcb*.

Uno pseudo-linguaggio funzionale puro non tipato

Sintassi della definizione di funzioni unarie:

$$
\begin{array}{c}
f(w_1) = \ldots corpo_1 \ldots \\
\ldots \\
f(w_n) = \ldots corpo_n \ldots
\end{array}
$$

- f è il nome della funzione che stiamo definendo
- w_i (i-esimo pattern) è una stringa sull'alfabeto formato da simboli terminali (costanti, costruttori) e da parametri formali (variabili)
- $corpo_i$ (i-esimo corpo di f) è un'espressione in pseudo-codice che può utilizzare
  - i parametri formali dell'i-esimo caso e tutte le costanti
  - espressioni if-then-else (sintassi C: $\ldots ? \ldots : \ldots$)
  - chiamate di funzione della forma $g(w)$ dove g è la funzione da chiamare e w è una stringa sull'alfabeto formato da simboli terminali e parametri formali di f

Uno pseudo-linguaggio funzionale puro non tipato

Quali funzioni *g* possono essere utilizzate?

- tutte quelle ovviamente implementabili e non interessanti ai fini dell'esercizio (p.e. operatori algebrici $+, *, =, \leq, \ldots$, operatori booleani $\&\&, ||, \ldots$, etc.)
- tutte quelle definite precedentemente nell'esercizio
- la *f* può invocare la stessa *f* (chiamata ricorsiva); in tal caso la *f* si dice funzione ricorsiva

IN QUESTO CORSO LE SOLE FUNZIONI RICORSIVE AMMESSE SARANNO QUELLE STRUTTURALMENTE RICORSIVE

Pattern matching

Sia ρ una stringa di terminali e ω una stringa di terminali e non terminali (variabili).

ρ fa match con ω se ottengo ρ da ω sostituendo a ogni non terminale $X_i \in \omega$ una sottostringa ρ_i di ρ.

Es.: *abba* matcha *aXa* sostituendo *bb* a X.
Es.: $3 + 4 * 5$ matcha $F_1 + F_2$ sostituendo 3 a F_1 e $4 * 5$ a F_2.
Es.: *abba* non matcha *bXb*.

Pattern matching

L'invocazione di una funzione avviene per pattern matching:

- Sia $f(\rho)$ la chiamata di funzione, dove ρ è una stringa di terminali
- Supponiamo che ci sia una e una sola dichiarazione $f(\omega_i) = corpo_i$ tale che ρ matchi ω_i
- La chiamata $f(\rho)$ viene riscritta in $corpo_i$ dopo aver sostituito a ogni parametro formale $X_j \in \omega$ la sottostringa ρ_j di ρ determinata durante il matching.

Esempio: data
$f([]) = 0$
$f(X :: L) = 1 + f(L)$

si ha $f(2 :: 3 :: []) \mapsto 1 + f(3 :: [])$
perchè $2 :: 3 :: []$ matcha $X :: L$ sostituendo a L $3 :: []$.

Uno pseudo-linguaggio funzionale puro non tipato

Esempi che non usano ricorsione
(dove $\{\langle, \text{“}, \text{”}, \rangle\}$ sono costruttori per le coppie e
$\{::\}$ è un costruttore che separa la testa dalla coda di una lista
non vuota, $\{[]\}$ rappresenta la lista vuota e $\bot$ un errore):

$$f(\langle x, y\rangle) = x + y$$

$$double(x) = f(\langle x, x\rangle)$$

$$max(\langle x, y\rangle) = \text{if } x < y \text{ then } y \text{ else } x$$

$$testa(hd :: tl) = hd$$
$$testa([]) = \bot$$

$$coda(hd :: tl) = tl$$
$$coda([]) = \bot$$

Uno pseudo-linguaggio funzionale puro non tipato

$f(\langle x, y \rangle) = x + y$
$double(x) = f(\langle x, x \rangle)$

Esempio di esecuzione:

$$double(4) \to f(\langle 4, 4 \rangle) \to 4 + 4 \to 8$$

Nel primo passaggio il parametro attuale 4 è stato sostituito al posto del parametro formale x nel corpo della *double*

Nel secondo passaggio l'input $\langle 4, 4 \rangle$ è stato confrontato con la stringa $\langle x, y \rangle$ ottenendo la sostituzione del primo 4 al posto di x e del secondo 4 al posto di y

Uno pseudo-linguaggio funzionale puro non tipato

$coda([]) = \bot$
$coda(hd :: tl) = tl$

Esempi di esecuzione:

$$coda(coda(1 :: [])) \to coda([]) \to \bot$$

Nel primo passaggio è stato selezionato il secondo corpo confrontando $1 :: []$ con i pattern $[]$ e $hd :: tl$. Solamente il primo faceva match.

Nel secondo passaggio è stato selezionato il primo corpo confrontando $[]$ con i pattern $[]$ e $hd :: tl$.

Uno pseudo-linguaggio funzionale puro non tipato

Cosa NON si può usare:

- assegnamenti alle variabili/array e/o altri side effect (allocazioni di memoria, I/O, . . .)
- cicli (for, while, repeat, . . .)
  inutili in assenza di side effect!

Uno pseudo-linguaggio funzionale puro non tipato

Terminologia:

- pseudo-linguaggio: non ci interessa fissare ulteriormente la sintassi, introdurre tipi di dati primitivi, definizioni di tipo, commenti, definizioni di funzioni ovviamente implementabili, etc.
- funzionale: le funzioni sono dati come gli altri, li potete prendere in input, dare in output, memorizzare in variabili, etc.
- puro: completamente privo di side effect e strutture dati mutabili
- non tipato: non imponiamo un sistema di tipi per prevenire staticamente (= prima di eseguire, durante la compilazione) errori stupidi

Uno pseudo-linguaggio funzionale puro non tipato

Potenza espressiva:

Se non ci restringessimo alla ricorsione strutturale (vedi dopo), qualunque programma esprimibile in un qualunque linguaggio di programmazione sarebbe esprimibile in questo pseudo-codice!
(Turing-completezza del linguaggio)

(ovviamente ignorando I/O, grafica, trasmissioni di rete, etc.)

Ricorsione strutturale: intuizione

- Le BNF imprimono alle stringhe una struttura ricorsiva
- In una struttura ricorsiva, un elemento o è atomico, o è composto a partire (non solo) da parti più piccole con la stessa struttura

Come si risolve un problema su una struttura dati ricorsiva (di dimensione finita, ma arbitraria)? I programmi sono tutti finiti!

Ricorsione strutturale:

- Nei casi atomici il problema è semplice e si risolve direttamente
- Nei casi composti, si risolve prima il problema sulle componenti (tramite ricorsione) e, una volta ottenute la risposte, si sintetizza la risposta per il caso composto senza osservare più i componenti $\Rightarrow$ uniformità della soluzione

Ricorsione strutturale

Poichè ogni stringa del linguaggio di una BNF ha una struttura ricorsiva, posso naturalmente definire delle funzioni ricorsive sulle stringhe del linguaggio.

Una funzione $f(\omega)$ dove ω è una stringa (formula) è definita per ricorsione strutturale sse

1. f considera tutte le possibili produzioni che definiscono ω una e una volta sola
2. per ogni produzione f si richiama ricorsivamente solamente sulle sottoformule immediate di ω

Esempio:

$$F ::= x \mid F + F \mid F * F$$

$$\begin{array}{lcl} size(x) & = & 1 \\ size(F_1 + F_2) & = & size(F_1) + size(F_2) \\ size(F_1 * F_2) & = & size(F_1) + size(F_2) \end{array}$$

Ricorsione strutturale

$$F ::= x \mid F + F \mid F * F$$

$$\begin{array}{lcl} size(x) & = & 1 \\ size(F_1 + F_2) & = & size(F_1) + size(F_2) \\ size(F_1 * F_2) & = & size(F_1) + size(F_2) \end{array}$$

Esempio di esecuzione:

$\quad size(x + ((x + x) * x))$
$\to size(x) + size((x + x) * x)$
$\to 1 + size(x + x) + size(x)$
$\to 1 + size(x) + size(x) + 1$
$\to 1 + 1 + 1 + 1$
$\to 4$

Ricorsione strutturale

$$X ::= \epsilon \mid aXa \mid bXb$$

Esempio di ricorsione strutturale corretta:

$$\begin{array}{lcl} length(\epsilon) & = & 0 \\ length(aXa) & = & 2 + length(X) \\ length(bXb) & = & 2 + length(X) \end{array}$$

Esempio di ricorsione strutturale errata (struttura errata):

$$\begin{array}{lcl} length(\epsilon) & = & 0 \\ length(aX) & = & 1 + length(X) \\ length(bX) & = & 1 + length(X) \end{array}$$

Ricorsione strutturale

$$X ::= \epsilon \mid aXa \mid bXb$$

Esempio di ricorsione strutturale corretta:

$$\begin{array}{lcl} uses_a(\epsilon) & = & false \\ uses_a(aXa) & = & true \\ uses_a(bXb) & = & uses_a(X) \end{array}$$

Esempio di ricorsione strutturale errata (manca una produzione):

$$\begin{array}{lcl} uses_b(aXa) & = & uses_b(X) \\ uses_b(bXb) & = & true \end{array}$$

Ricorsione strutturale

$$X ::= \epsilon \mid aXa \mid bXb$$

Esempio di ricorsione strutturale corretta:

$$\begin{array}{rcl} all_a(\epsilon) & = & true \\ all_a(aXa) & = & all_a(X) \\ all_a(bXb) & = & false \end{array}$$

Esempio di ricorsione strutturale errata (chiamate ricorsive non sulle sottoformule):

$$\begin{array}{rcl} f(\epsilon) & = & true \\ f(aXa) & = & f(bXb) \\ f(bXb) & = & f(aXa) \end{array}$$

Ricorsione strutturale

$$X ::= \epsilon \mid aXa \mid bXb$$

Esempio di ricorsione strutturale errata (struttura errata):

$$\begin{array}{lcl} middle_{bb}(\epsilon) & = & true \\ middle_{bb}(aa) & = & false \\ middle_{bb}(aXa) & = & middle_{bb}(X) \\ middle_{bb}(bXb) & = & middle_{bb}(X) \end{array}$$

Ricorsione strutturale

Teorema: tutte le funzioni definite per ricorsione strutturale convergono su ogni input.

Dimostrazione: a ogni chiamata ricorsiva, il numero di espansioni di non terminali necessarie per ottenere la stringa in input cala di 1 e, prima o poi, arriva a 0.

Teorema: ci sono problemi risolvibili con la ricorsione generale (= non strutturale) ma non con la ricorsione strutturale

Tuttavia: nella maggior parte dei casi, se la vostra ricorsione non è strutturale, o non state usando la struttura (struttura dati) giusta, o la funzione è bacata o il codice è semplificabile.

Funzioni *n*-arie

Per comodità estendiamo il linguaggio di programmazione al caso di funzioni *n*-arie, ovvero che prendono più argomenti.

$$
\begin{array}{c}
f(w_1^1, \ldots, w_1^n) = \ldots corpo_1 \ldots \\
\ldots \\
f(w_m^1, \ldots, w_m^n) = \ldots corpo_m \ldots
\end{array}
$$

Una funzione *n*-aria è scritta per ricorsione strutturale sul primo argomento quando soddisfa i vincoli della ricorsione strutturale imposti solamente sul primo argomento.

Funzioni *n*-arie e ricorsione strutturale

Esempio: questa funzione è per ricorsione strutturale sulla lista che è il primo argomento.

$$\begin{array}{lcl} size(X :: L, A) & = & size(L, A+1) \\ size([], A) & = & A \end{array}$$

La chiamata $size(1 :: 2 :: 3 :: [], A)$ calcola 3 + A
(la lunghezza della lista $1 :: 2 :: 3 :: []$ più A).
Per $A = 0$ calcola la lunghezza della lista.

Il parametro A si chiama accumulatore perchè accumula informazioni sulla parte di struttura dati già visitata. In questo caso, accumula la sua lunghezza.

Grazie agli accumulatori si può evitare di memorizzare informazioni in variabili imperative.

Ricapitoliamo

1. BNF (+ precedenza e associatività): metodo per definire la grammatica di un linguaggio
2. La BNF induce una struttura ricorsiva sulle stringhe del linguaggio
3. La struttura ricorsiva induce una classe di funzioni ricorsive totali (= convergenti su ogni input). Tale ricorsione si dice strutturale.

Induzione strutturale

Come si dimostra che una funzione definita per ricorsione strutturale gode di una certa proprietà? Usando l'induzione strutturale!

Sia P una proprietà che vogliamo dimostrare valere su tutte le stringhe ω di un linguaggio generato da una BNF. La dimostrazione può essere data in questo modo:

1. Una sotto-dimostrazione per ogni produzione che genera ω
2. In ogni sotto-dimostrazione possiamo assumere che P già valga su tutte le sotto-formule immediate di ω (ipotesi induttive)

Intuizione: stiamo definendo la dimostrazione per ricorsione strutturale. Le ipotesi induttive sono le chiamate ricorsive.

Induzione strutturale

$$X ::= \epsilon \mid aXa \mid bXb$$

$$\begin{array}{lcl} length(\epsilon) & = & 0 \\ length(aXa) & = & 2 + length(X) \\ length(bXb) & = & 2 + length(X) \end{array}$$

Esempio di induzione strutturale corretta:

Vogliamo dimostrare che per ogni X la $length(X)$ non è dispari.

Caso ϵ: $length(\epsilon) = 0$ e 0 non è dispari.

Caso aXa:

Per ipotesi induttiva $length(X)$ non è dispari.

Si ha $length(aXa) = 2 + length(X)$ che non è dispari poichè altrimenti $length(X)$ sarebbe dispari, ma sappiamo per ipotesi induttiva che non lo è.

Caso bXb: analogo

Induzione strutturale

Perchè l'induzione strutturale funziona?

Riprendiamo l'esempio e consideriamo *abba*.

$length(abba) = 2 + length(bb) = 2 + (2 + length(\epsilon)) = 2 + (2 + 0)$

$length(abba)$ non è dispari perchè lo abbiamo dimostrato assumendo che $length(bb)$ non sia dispari che è vero perchè $length(bb)$ non è dispari perchè lo abbiamo dimostrato assumendo che $length(\epsilon)$ non sia dispari che è vero perchè $length(\epsilon)$ non è dispari perchè lo abbiamo dimostrato.

La ricorsione strutturale funziona per lo stesso motivo per il quale la ricorsione strutturale converge su ogni input!

Conclusioni

1. BNF (+ precedenza e associatività): metodo per definire la grammatica di un linguaggio
2. La BNF induce una struttura ricorsiva sulle stringhe del linguaggio
3. La struttura ricorsiva induce una classe di funzioni ricorsive totali (= convergenti su ogni input). Tale ricorsione si dice strutturale.
4. La struttura ricorsiva induce una classe di dimostrazioni ricorsive corrette. Tali dimostrazioni si dicono per ricorsione strutturale.

La sintassi della logica proposizionale

Formule della logica proposizionale:

$$F ::= \bot \mid \top \mid A \mid B \mid \ \ldots \ \mid \neg F \mid F \wedge F \mid F \vee F \mid F \Rightarrow F$$

Semantica intuitiva:

$\bot$ denota la falsità

$\top$ denota la verità

A, B, ... denotano un valore di verità sconosciuto/non determinato (dipende dal mondo)

$\neg F$ è la negazione di F ("not F")

$F_1 \wedge F_2$ è la congiunzione di due formule ("F_1 e F_2")

$F_1 \vee F_2$ è la disgiunzione inclusiva di due formule ("F_1 o F_2")

$F_1 \Rightarrow F_2$ è l'implicazione materiale di due formule ("se F_1 allora F_2")

La sintassi della logica proposizionale

Formule della logica proposizionale:

$$F ::= \bot \mid \top \mid A \mid B \mid \ldots \mid \neg F \mid F \wedge F \mid F \vee F \mid F \Rightarrow F$$

Rendiamo la sintassi non ambigua:

- Precedenze: $\neg > \wedge > \vee > \Rightarrow$
- Associatività: a destra per tutti gli operatori

Esempi:

- $\neg A \wedge B \vee \top \Rightarrow C$ si legge $(((\neg A) \wedge B) \vee \top) \Rightarrow C$
- $A \Rightarrow B \Rightarrow C$ si legge $A \Rightarrow (B \Rightarrow C)$ e non $(A \Rightarrow B) \Rightarrow C$!

Logica proposizionale

La logica proposizionale studia solamente le connotazioni che denotano valori di verità.

Ovvero ogni (sotto)formula è una sentenza.

La scelta dei connettivi al momento sembra arbitraria (perchè quelli e non altri? perchè tutti quelli?) Verrà chiarita in seguito (tempo permettendo).

La semantica delle formule verrà chiarita formalmente in seguito.

In seguito nel corso vedremo una logica più espressiva con connotazioni che non sono sentenze e con quantificatori (per ogni, esiste, ...).

Formalizzazione

Con formalizzazione di una frase in linguaggio naturale si intende trovare la formula logica che meglio approssima la frase. Esempi:

- Se 2 + 2 fa 5 allora io sono una carriola.
  Formalizzazione: $A \Rightarrow B$ dove
  A sta per "2+2 fa 5"
  B sta per "io sono una carriola"
- Non è vero che quando fa caldo bisogna accendere il condizionatore.
  Formalizzazione: $\neg(A \Rightarrow B)$ dove A sta per "fa caldo"
  B sta per "bisogna accendere il condizionatore"

Formalizzazione

Difficoltà nella formalizzazione:

1. Connotazioni diverse per gli stessi connettivi. Esempi:
   - "Se A allora B", "A implica B", "B se A", "B quando A", "quando A, B", "A è condizione sufficiente per B", "B è condizione necessaria per A", ...
   - "A e B", "A ma B", "A nonostante B", ...
2. Sinonimi e contrari. Esempio:
   "se Mario è acculturato allora oggi c'è bel tempo", "oggi splende il sole e Mario è ignorante" si formalizza come
   $M \Rightarrow B$, $B \wedge \neg M$

Esercizi

Definire per ricorsione strutturale sulle formule della logica proposizionale alcune funzioni per

1. calcolare il numero di simboli in una formula
2. calcolare l'altezza di una formula (più lungo cammino radice-foglia dell'albero sintattico)
3. decidere se una formula non contiene negazioni
4. decidere se una formula non contiene variabili proposizionali ($A, B, \ldots$)
5. decidere se una formula è bilanciata, ovvero l'altezza delle due sotto-formule di ogni connettivo binario è la stessa

Esercizi

Dimostrare per induzione strutturale sulle formule della logica proposizionale che

1. l'altezza di una formula è sempre minore o uguale al numero di suoi simboli
2. se la formula non contiene connettivi binari, allora l'altezza e il numero di simboli coincidono
3. se la formula è bilanciata e non contiene negazioni, allora il numero di simboli è $2^h - 1$ dove h è l'altezza dell'albero