INDICE

Calcolo degli enunciati

Propositional Logic: Syntax

Definizione**:** un **alfabeto** ∑ consiste in un insieme finito o infinito contabile. Gli elementi di questo insieme sono chiamati **simboli.**

Si assume che Λ ∉ ∑ dove Λ sta per “Parola Vuota”

L’insieme delle **parole** (o **stringhe**) in ∑ è denotato da ∑* ed è definito come:

1. Λ ∈ ∑*
2. se ω ∈ ∑*, a ∈ ∑ allora aω ∈ ∑*
3. non ci sono altre parole in ∑*

Quindi ∑ è l’insieme dei simboli, mentre ∑* è l’insieme delle parole formate dai simboli.

Per esempio: se abbiamo ∑ = {1,2} allora possiamo avere le parole: Λ, 1Λ, 2Λ, 11Λ, 22Λ, 12Λ, 21Λ, 111Λ, …

Alfabeto della logica proposizionale

L’alfabeto della logica proposizionale consiste di:

- un insieme infinito ma contabile di **variabili proposizionali** R = {p1, p2, …}
- L’insieme di connettivi {¬/1, ^/2, v/2 →/2,↔/2}
- I caratteri speciali “)” e “(“

Per variabile proposizionale una variabile come quelle usate in matematica per rappresentare numeri o espressioni. Nella logica queste variabili rappresentano o una formula proposizionale o un valore di vero o falso.

Quando l’alfabeto è in funzione bigettiva coi numeri naturali è possibile eliminare l’insieme infinito generando una grammatica che con un insieme finito di simboli può generare infinite combinazioni associabili ai numeri naturali.

Linguaggio Proposizionale

Una formula proposizionale atomica (chiamata **atomo**) è una **variabile proposizionale**.

L’insieme di formule proposizionali L(R) è il più piccolo insieme con le seguenti proprietà:

1. Se F è una formula proposizionale, allora F ∈ L(R)
2. Se F ∈ L(R), allora ¬F ∈ L(R)
3. Se °/2 è un generico connettivo binario e F,G ∈ L(R), allora (F ° G) ∈ L(R)

Induzione Strutturale

Slide “Concetti di matematica di base” da 2 a 5

Voglio dimostrare che la proprietà P è vera per tutti i numeri:

- Affermo che è vera per P(0)
- Premessa: è vero che P(x), bisogna dimostrare che è vero P(x+1)
- Quindi $\forall$y, P(y) è vera

Ciò che significa in termini informatici?

- So che P*(0) è vera e voglio dimostrare che P* vale su tutti gli $\mathbb{N}$
- Ho un programma che, data la dimostrazione di un oggetto, mi da la dimostrazione dell’oggetto successivo
- Quindi P*(x) — programma(x) → P*(x+1)

L’induzione è un processo ricorsivo. Una volta definito il passo i_0, può definire il passo i_1 basandosi su i_0. Per definire i_n si basa su i_{n-1}, i_{n-2}, …,i_0!

Come possiamo dimostrare una proprietà su un insieme di formule?

Principio di induzione strutturale:

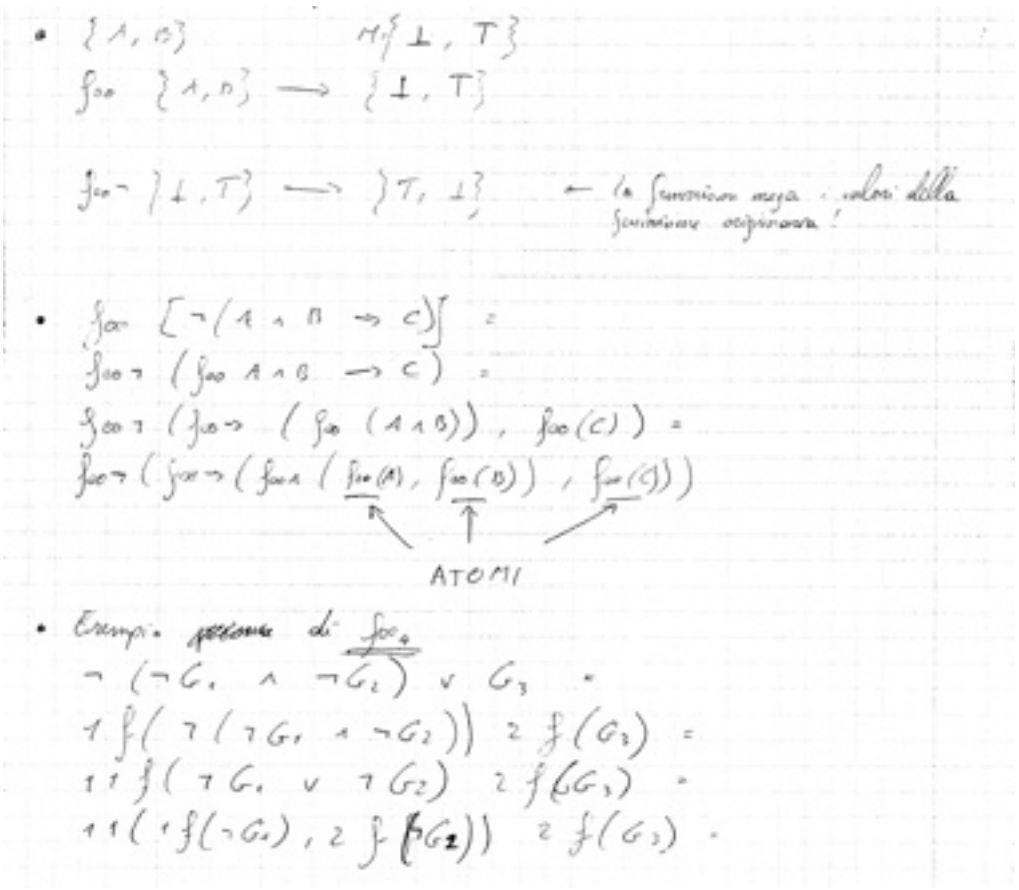
Ogni atomo ha la proprietà E se:

- F ha la proprietà E, allora ¬F ha E
- °/2 è un connettivo binario e F e G hanno la proprietà E, alla anche (F ° G) ha E

Ricorsione strutturale

Registrazione del 09/10/2009 - 10:13 I Slide dalla ..

Esempi alla lavagna



Sottoformule

Registrazione al minuto 28

Semantica

Registrazione al minuto 31

Trovare la semantica di una formula vuol dire trovare il suo valore (nella logica: vero o falso), trovando il significato di ogni connettivo.

Struttura delle tavole di verità

La struttura delle tavole di verità consiste in un insieme di valori di verità ($\bot$, $\top$) e di funzioni definite su di essi.

Funzioni unarie

- Negazione $\neg^* = \neg^*(\top) = \bot$
- Identità
- Proiezione su $\top$ e $\bot$

Funzioni Binarie

- Congiunzione ^*/2
- Disgiunzione v*/2
- Implicazione →*/2
- NAND ↑*/2
- NOR ↓*/2
- Equivalenza ↔*/2

NB: L'asterisco accanto al simbolo della funzione, indica che ci stiamo riferendo al significato semantico della funzione; alla specifica interpretazione che è stato deciso di dare al simbolo.

La freccia presente nella formula è un elemento dell'alfabeto di partenza (A → B);

La freccia asteriscata, invece, sta ad identificare la funzione che io a quella freccia assegno, così come ad A assegno un valore o una sottoformula (A →* B).

La semantica dei nostri connettivi rappresentati dai simboli dell'alfabeto {→, ↔, ^, v, … } è la seguente:

		$\wedge^*$	$\vee^*$	$\rightarrow^*$	$\leftarrow^*$	$\uparrow^*$	$\downarrow^*$	$\not\rightarrow^*$	$\not\leftarrow^*$
⊤	⊤	⊤	⊤	⊤	⊤	⊥	⊥	⊥	⊥
⊤	⊥	⊥	⊤	⊥	⊤	⊤	⊥	⊤	⊥
⊥	⊤	⊥	⊤	⊤	⊥	⊤	⊥	⊥	⊤
⊥	⊥	⊥	⊥	⊤	⊤	⊤	⊤	⊥	⊥

		$\leftrightarrow^*$	$\not\leftrightarrow^*$						
⊤	⊤	⊤	⊥	⊤	⊥	⊤	⊥	⊤	⊥
⊤	⊥	⊥	⊤	⊤	⊥	⊤	⊥	⊥	⊤
⊥	⊤	⊥	⊤	⊤	⊥	⊥	⊤	⊤	⊥
⊥	⊥	⊤	⊥	⊤	⊥	⊥	⊤	⊥	⊤

Interpretazione e modelli

Interpretare una formula significa dare un significato ai simboli che la rappresentano.

Definizione: Un'interpretazione (proposizionale) I = (W , $\bullet^I$) consiste in un insieme W ed un'interpretazione $\bullet^I$: L(R) → W con

$$[F]^I = \begin{cases} w \in \mathcal{W} & \text{if } F \in \mathcal{R}, \\ \neg^*[G]^I & \text{if } F \text{ is of the form } \neg G, \\ ([G_1]^I \circ^* [G_2]^I) & \text{if } F \text{ is of the form } (G_1 \circ G_2). \end{cases}$$

- Un'interpretazione I = (W , $\bullet^I$) è unicamente definita specificando come $\bullet^I$ agisce sulle variabili proposizionali
- Denotiamo con A un "atomo". Potremo avere interpretazioni del tipo I = (W , $\bullet^I$) con {A | $[A]^I = \top$ }
- Si può scrivere F^I anziché $[F]^I$
- I (possibilmente indicizzato) denota un'interpretazione

Soddisfacibilità di una formula

La formula F si può dire:

- **Soddisfacibile:** se c'è almeno un'interpretazione per cui F è vera
- **Tautologia (valida)**: se per tutte le interpretazioni F è vera
- **Falsificabile**: se c'è almeno un'interpretazione per cui F è false
- **Insoddisfacibile:** se tutte le interpretazioni di F sono false

Modelli

Definizione: Un'interpretazione I = (W, $\bullet^I$) è detta modello per una formula proposizionale F, in simboli I $\models$ F, se F^I = T

Un'interpretazione è modello se tramite essa F da valore vero.

Nota che $\models$ non è un simbolo del linguaggio (detto linguaggio oggetto), ma è un simbolo che noi usiamo per studiare il linguaggio (usando un metalinguaggio)

Validità e insoddisfacibilità

Teorema: Una formula proposizionale F è valida ($\models$ F) se e solo se ¬F è insoddisfacibile.

$\models$ F sse tutte le interpretazioni sono modelli per F
sse nessuna interpretazione è modello per ¬F
sse ¬F è insoddisfacibile

Soddisfacibilità di un insieme di formule

Definizione: Sia G un insieme di formule proposizionali

- G è soddisfacibile se esiste un'interpretazione per cui ogni F $\in$ G = $\top$
- Questa interpretazione I è detta modello per l'insieme di formule G (I $\models$ G) dato che per ogni F $\in$ G si ha che I $\models$ F

Conseguenze logiche

Una formula proposizionale F è una **conseguenza** di un insieme di formule G, in simboli **G ⊨ F**, se e solo se per ogni interpretazione I abbiamo che: **I ⊨ F sse I ⊨ G**

Data l'interpretazione, se è vero l'insieme di partenza, risulta che è vero l'insieme che ne segue.

La conseguenza logica si indica col simbolo ⊨

Si dice che, dato G un insieme di Formule Proposizionali ed F una formula proposizionale, **G ⊨ F** (F è conseguenza per G) se e solo se **G → F è valida (una tautologia)**

ES:

- **{(p v q)} ⊨ q** **<-- falsa**

p	q	p v q	→	q
v	v	v	v	v
f	v	v	--- f ---	f
v	f	v	v	v
f	f	f	v	f

- **{p, (p → q)} ⊨ q** **<--- Valida**
  NB: in questo caso ponendo F_1 = p e F_2 = (p → q) prima di verificare la conseguenza devo calcolare F_1 ^ F_2

p	q	p → q	F_1 ^ F_2	→	q
v	v	v	v	v	v
f	v	v	f	v	f
v	f	v	v	v	v
f	f	f	f	v	f

NB: Ricorda che nell'esercizio sulla slide la prima è vera, la seconda è falsa e la terza è sempre vera (crea un assurdo)

Tavole di Verità

Se F è una formula e T la tavola di verità corrispondente, allora F è:

- **soddisfacibile** se T contiene una riga con T nell'ultima colonna
- **valida** se tutte le righe in T hanno T nell'ultima colonna
- **falsificabile** se T contiene una riga con ⊥ nella colonna finale
- **insoddisfacibile** se tutte le righe T hanno ⊥ nell'ultima colonna

Le tavole di verità sono un sistema che si ingrandisce in maniera esponenziale, quindi è molto pesante da calcolare.

C'è un sistema alternativo alle tabelle di verità, più rapido per la verifica di una formula.

Equivalenza Semantica

Definizione: Due formule proposizionali F e G si dicono semanticamente equivalenti (F ≡ G) sse per tutte le interpretazioni I si ha che: **I ⊨ F se e solo se I ⊨ G**

NB: Non confondere con la conseguenza proposizionale! Qui F e G sono Formule proposizionali. Nel caso della conseguenza logica, con G si intendeva un insieme di formule!

L'equivalenza semantica è una relazione che possiede le seguenti proprietà:

- **Riflessiva**: F ≡ F
- **Simmetrica**: se F ≡ G allora G ≡ F
- **Transitiva**: se F ≡ G e G ≡ H allora H ≡ F

Equivalenze Note

$(F \wedge F)$	$\equiv$	F	
$(F \vee F)$	$\equiv$	F	idempotency
$(F \wedge G)$	$\equiv$	$(G \wedge F)$	
$(F \vee G)$	$\equiv$	$(G \vee F)$	commutativity
$((F \wedge G) \wedge H)$	$\equiv$	$(F \wedge (G \wedge H))$	
$((F \vee G) \vee H)$	$\equiv$	$(F \vee (G \vee H))$	associativity
$((F \wedge G) \vee F)$	$\equiv$	F	
$((F \vee G) \wedge F)$	$\equiv$	F	absorption
$(F \wedge (G \vee H))$	$\equiv$	$((F \wedge G) \vee (F \wedge H))$	
$(F \vee (G \wedge H))$	$\equiv$	$((F \vee G) \wedge (F \vee H))$	distributivity

$\neg\neg F$	$\equiv$	F	double negation
$\neg(F \wedge G)$	$\equiv$	$(\neg F \vee \neg G)$	
$\neg(F \vee G)$	$\equiv$	$(\neg F \wedge \neg G)$	de Morgan
$(F \vee G)$	$\equiv$	F, if F is valid	
$(F \wedge G)$	$\equiv$	G, if F is valid	tautology
$(F \vee G)$	$\equiv$	G, if F is unsatisfiable	
$(F \wedge G)$	$\equiv$	F, if F is unsatisfiable	unsatisfiability
$(F \leftrightarrow G)$	$\equiv$	$((F \wedge G) \vee (\neg G \wedge \neg F))$	equivalence
$(F \rightarrow G)$	$\equiv$	$(\neg F \vee G)$	implication

Posizioni

Con la seguente formula è possibile creare un insieme delle parti P(F) contenente stringhe formate dai simboli {1,2} più il simbolo Λ che indica la parola vuota. Queste stringhe rappresentano le posizioni degli atomi all'interno di una formula proposizionale F:

$$foo_1(F) = \{\Lambda\} \cup \begin{cases} \emptyset & \text{if } F \text{ is an atom,} \\ \{1\pi \mid \pi \in foo_1(G)\} & \text{if } F \text{ is of form } \neg G, \\ \{1\pi_1 \mid \pi_1 \in foo_1(G_1)\} \cup \{2\pi_2 \mid \pi_2 \in foo_1(G_2)\} & \\ & \text{if } F \text{ is of the form } (G_1 \circ G_2). \end{cases}$$

NB: il pi greco indica "il resto della stringa". 1π, indica "1" + la stringa rimanente.

Sostituzioni

La sottoformula di F alla posizione π ∈ P (F), in simboli espressa come **F[π],** è definita da:

- F[Λ] = F
- F[1π] = G[π], se F è nella forma ¬G
- F[iπ] = G_i [π], se F è nella forma (G_1 • G_2) e i ∈ {1,2}

La sostituzione di un insieme di sottoformule di F alla posizione π ∈ P(F) con H, in simboli **F[π → H],** è definita da:

- F[Λ → H] = H
- F[1π → H] = ¬(G [π → H]), se F è nella forma ¬G
- F[1π → H] = (G_1 [π → H] • G_2), se F è nella forma (G_1 • G_2)
- F[2π → H] = (G_1 • G_2 [π → H]), se F è nella forma (G_1 • G_2)

Esempio:

Avendo la formula F = ¬ (A ^ B –> C), dico che F ≡ F [112 –> H]. Avrò:

¬ (A ^ B → C) [112 → H]	F è nella forma ¬G. Ho 1π, quindi escludo la negazione
¬ (A ^ B → C [12 → H])	F è nella forma (G1 → G2). Ho 1π, quindi entro in G1
¬ (A ^ B [2 → H] → C)	F è nella forma (G1 ^ G2). Ho 2π, quindi entro in G2
¬ (A ^ B [Λ → H] → C)	Ho raggiunto il simbolo Λ, quindi sostituisco l'atomo B
¬ (A ^ H → C)	

Teorema della sostituzione

Teorema (del rimpiazzamento):

- **Theorem 3.18 (Replacement Theorem)**
  Let $F \in \mathcal{L}(\mathcal{R})$, $\pi \in \mathcal{P}(F)$, $F\lceil\pi\rceil = G$ and $G \equiv H$.
  Then, $F \equiv F\lceil\pi \mapsto H\rceil$.
- **Proof** Structural induction on π.
- **Induction basis** Let $\pi = \Lambda$.
  - $F = F\lceil\Lambda\rceil = G \equiv H = F\lceil\Lambda \mapsto H\rceil$. ✓
- **Induction hypothesis (IH)** The theorem holds for π'.
- **Induction step** Let $\pi = i\pi'$. We distinguish two cases:

  $i = 1$ $\pi = 1\pi' \in \mathcal{P}(F)$, hence F must be otf $\neg F'$ or $(G_1 \circ G_2)$.

  $\neg F'$ $F\lceil 1\pi' \mapsto H\rceil = (\neg F')\lceil 1\pi' \mapsto H\rceil = \neg(F'\lceil\pi' \mapsto H\rceil)$.
  From IH we conclude: $F' \equiv F'\lceil\pi' \mapsto H\rceil$
  Hence, for all interpretations I we find:

$[F]^I$	$=$	$[\neg F']^I$	Assumption
	$=$	$\neg^*[F']^I$	Def I
	$=$	$\neg^*[F'\lceil\pi' \mapsto H\rceil]^I$	IH and Def $\equiv$
	$=$	$[\neg(F'\lceil\pi' \mapsto H\rceil)]^I$	Def I
	$=$	$[(\neg F')\lceil 1\pi' \mapsto H\rceil]^I$	Def replacement
	$=$	$[F\lceil 1\pi' \mapsto H\rceil]^I$	Assumption

  Hence, $F \equiv F\lceil 1\pi' \mapsto H\rceil$ ✓

  $(G_1 \circ G_2)$ analogously

  $i = 2$ analogously qed

Agreement

- If $\pi \notin \mathcal{P}(F)$ then $F\lceil\pi \mapsto H\rceil := F$.
- Let $F\lceil\pi\rceil = G$.
  If it is obvious from the context which subformula G is to be replaced,
  then we write $F\lceil G \mapsto H\rceil$ instead of $F\lceil\pi \mapsto H\rceil$.
- Theorem 3.18 allows to replace all occurrences of the connectives $\leftrightarrow$ and $\rightarrow$.
  - Wlog we consider only formulas over $\mathcal{R} \cup \{\neg, \wedge, \vee, (,)\}$.

Forma Normale Negata

Definizione: Una formula proposizionale F è in **forma normale negata** se tutte le negazioni ¬ presenti in F appaiono direttamente prima delle variabili proposizionali

Algoritmo per la conversione: sia data F. Finché F non è in forma normale negata allora fai:

- Prendi una sottoformula di F avendo la forma ¬G con G ∉ R
- se ¬G = ¬¬H allora sostituisco ¬¬H con H
- se ¬G=¬(G1 ^ G2) allora sostituisco ¬(G1^G2) con (¬G1 v ¬G2)
- se ¬G=¬(G1 v G2) allora sostituisco ¬(G1vG2) con (¬G1 ^ ¬G2)

$$\frac{\neg\neg H}{H} \qquad \frac{\neg(G_1 \wedge G_2)}{(\neg G_1 \vee \neg G_2)} \qquad \frac{\neg(G_1 \vee G_2)}{(\neg G_1 \wedge \neg G_2)}$$

Forma Clausale

Definizione: una *letterale (litera)* è una variabile proposizionale o una variabile proposizionale negata.

- L (possible indexed) denotes a literal.
- Generalized disjunction

$$[F_1, \ldots, F_n] = (\ldots((F_1 \vee F_2) \vee F_3) \vee \ldots \vee F_n)$$

- Generalized conjunction

$$\langle F_1, \ldots, F_n \rangle = (\ldots((F_1 \wedge F_2) \wedge F_3) \wedge \ldots \wedge F_n)$$

- Observe $[F] = F$ and $\langle F \rangle = F$. Hence, $[F] \equiv F$ and $\langle F \rangle \equiv F$.
- Empty generalized disjunction: $[\,]$ with $[\,]^I = \bot$ for all I.
- Empty generalized conjunction: $\langle\rangle$ with $\langle\rangle^I = \top$ for all I.
- Observe $(G \vee [\,]) \equiv G$ and $(G \wedge \langle\rangle) \equiv G$.

- A clause is a generalized disjunction $[L_1, \ldots, L_n]$, $n \geq 0$, where every L_i, $1 \leq i \leq n$, is a literal.
- A dual clause is a generalized conjunction $\langle L_1, \ldots, L_n \rangle$, $n \geq 0$, where every L_i, $1 \leq i \leq n$, is a literal.
- A formula is in conjunctive normal form, or clause form, iff it is of the form $\langle C_1, \ldots, C_m \rangle$, $m \geq 0$, and if every C_j, $1 \leq j \leq m$, is a clause.
- A formula is in disjunctive normal form, or dual clause form, iff it is of the form $[C_1, \ldots, C_m]$, $m \geq 0$, and if every C_j, $1 \leq j \leq m$, is a dual clause.

[L1 v L2 v … v Ln] <-- Clausola (indicata con C)

[L1 ^ L2 ^ … ^ Ln] <-- Doppia Clausola (DC)

Forma Normale Congiuntiva o Forma Clausale: <C1 ^ C2 ^ … ^ Cn>

Forma Normale Disgiuntiva o Doppia forma Clausale: [DC1 v DC2 v … v DCn]

Algoritmo di trasformazione

registarzione del 26/10/2009 9.14

NB: Tutte queste regole di trasformazione possono portare ad un cammino di risoluzione che non è detto risolva la formula. A seconda del cammino che seguo posso o no trovare la soluzione.
Il metodo migliore è l'intuito. Se pesco un cammino che sembra non portare ad una soluzione, allora torno indietro e ne scelgo un altro.
In pratica si va avanti a tentativi.

NB2: Quando ho una soluzione del tipo D1 | D2, non posso interpretare | come un and. Devo elaborare D1 e D2 in maniera del tutto separata, non posso "chiuderli".

- Input: A propositional formula F.
  Output: A formula, which is in conjunctive normal form and is equivalent to F.
  $G := \langle [F] \rangle$. (G is a conjunction of disjunctions.)
  While G is not in conjunctive normal form do:
  Select a non-clausal element H from G.
  Select a non-literal element K from H.
  Apply the rule among the following ones which is applicable.

$$\frac{\neg\neg D}{D} \quad \frac{(D_1 \wedge D_2)}{D_1 \mid D_2} \quad \frac{\neg(D_1 \wedge D_2)}{\neg D_1, \neg D_2} \quad \frac{(D_1 \vee D_2)}{D_1, D_2} \quad \frac{\neg(D_1 \vee D_2)}{\neg D_1 \mid \neg D_2}$$

- A rule $\frac{D}{D'}$ is applicable to K if K is of the form D.
  If applied, then K is replaced by D'.
- A rule $\frac{D}{D_1 \mid D_2}$ is applicable to K if K is of the form D.
  If applied, H is replaced by two disjunctions:
  The first one is obtained from H by replacing the occurrence of D by D_1.
  The second one is obtained from H by replacing the occurrence of D by D_2.

- Example

$$\langle [\neg(p \vee (\neg(p \wedge q) \wedge \neg r))] \rangle$$
$$\langle [\neg p], [\neg(\neg(p \wedge q) \wedge \neg r)] \rangle$$
$$\langle [\neg p], [\neg\neg(p \wedge q), \neg\neg r] \rangle$$
$$\langle [\neg p], [(p \wedge q), \neg\neg r] \rangle$$
$$\langle [\neg p], [(p \wedge q), r] \rangle$$
$$\langle [\neg p], [p, r], [q, r] \rangle$$

Metodo di dimostrazione

Se ho un interpretazione che mi rende vere tutte le formule F1, F2, … Fn, allora è vera anche F:

$$\{F1, \ldots, Fn\} \models F$$

(vedi Conseguenze Logica)

sse (<F1,…,Fn> → F) è valida

Per provare questa asserzione, nego la formula e dimostro che sia insoddisfacibile:

¬(<F1, …,Fn> -> F) ≡
<F1, … Fn, ¬F>

che è insoddisfacibile

Risoluzione

Una formula scritta in forma di clausola non è soddisfacibile se

- contiene la clausola vuota (la semantica della clausola vuota è falso).
  Avendo una formula nella forma {F1,..,Fn} ⊨ F e, tramite un procedimento che la semplifica, ottengo una clausola vuota, quindi il falso, allora la formula non è soddisfacibile.
- Se invece la formula soltanto contiene la clausola vuota, vuol dire anche in quel caso che la formula è insoddisfacibile.
- Se invece semplificando raggiungo solo atomi, allora è soddisfacibile

Esempio:

Voglio provare che A^B → A è vera. Quindi dimostro che la sua negazione è insoddisfacibile:

¬ (A ^ B → A)
< A ^ B, ¬ A >

- Trasformo in Forma Clausale:

< [A, ¬A] , [B, ¬A] >

- Elimino tutte le occorrenze di A e ¬ A e ottengo :

< [ ], [B] >

- Quindi ho trovato la clausa vuota, il che significa che la negazione della mia formula è Insoddisfacibile e che quindi la mia formula è Vera.

Esempio:

- Some Facts
  - K_1 If it is hot and humid, then it will rain.
  - K_2 If it is humid, then it is hot.
  - K_3 It is humid now.
- Question
  - K_4 Will it rain?
- Introducing propositional variables
  - p it is hot
  - q it is humid
  - r it will rain
- Formalization
  - F_1 $((p \wedge q) \rightarrow r)$
  - F_2 $(q \rightarrow p)$
  - F_3 q
  - F_4 r

$$\{((p \wedge q) \rightarrow r),\ (q \rightarrow p),\ q\} \models r$$
iff
$$(\langle((p \wedge q) \rightarrow r),\ (q \rightarrow p),\ q\rangle \rightarrow r) \text{ is valid}$$
iff
$$\neg(\langle((p \wedge q) \rightarrow r),\ (q \rightarrow p),\ q\rangle \rightarrow r) \text{ is unsatisfiable}$$
iff
$$\langle((p \wedge q) \rightarrow r),\ (q \rightarrow p),\ q,\ \neg r\rangle \text{ is unsatisfiable}$$
iff
$$\langle[\neg p, \neg q, r],\ [\neg q, p],\ [q],\ [\neg r]\rangle \text{ is unsatisfiable}$$

Da qui posso semplificare ¬p e p avendo:

<[¬q, r], [¬q], [q], [¬r]>

Allo stesso modo posso semplificare ¬q e q:

<[r], [¬r]>

E continuando

< [ ] >

che è insoddisfacibile

NB: Guarda il seguito per capire perché si semplifica così

Regole di risoluzione

Dato che nelle formule ho A e ¬A, capisco che A è inutile alla risoluzione del problema, quindi lo elimino.

1. Rimuovere tutte le clausole di A da C1
2. Rimuovere tutte le clausole di ¬A da C2
3. Combiniamo le clausole ottenute con questo metodo disgiuntivo ottenendo una clausa C che deriva dalla combinazione di C1 e C2: C = [C1,C2]

Esempio:

```
          A ^ B → A
  ————————————————————————————
        ¬ (A ^ B → A )
        < A ^ B , ¬A >
```

Qui le formule si dividono in 2:

```
A ¬A          B ¬A
[ ]           [ B ]
```

Elimino A ¬A e quindi ottengo la clausola vuota. Da ciò ottengo che:

```
        ¬ (A ^ B → A )
```

è insoddisfacibile

Esempio:

```
          A v B → A
  ————————————————————————————
        ¬( A v B → A )
        < A v B , ¬A >
          A, B, ¬A
            B
```

Quindi ¬(A v B → A) è falsificabile

Ascolta l'esempio alle 9.30

Derivazione e confutazione

Definizione: Sia F = < C1,…, Cn> una formula in forma clausale

1. La sequenza (Ci | 1 ≤ i ≤ n) è una **derivazione risolutiva** per F
2. Se (Ci | 1 <= i <= m) è una derivazione risolutiva per F e Cm+1 è ottenuta attraverso l'applicazione delle regole di risoluzione a due elementi presi da (Ci | 1 <= i <= m), allora (Ci | 1 <= i <= m+1) è una derivazione risolutiva per F
3. Una derivazione risolutiva per F che contiene la clausola vuota è chiamata **risoluzione confutativa** per F.

- F può contenere [ ]
- La presenza di una [ ] è sufficiente per considerare una risoluzione confutativa
- Possiamo assumere che [ ] è l'ultima clausola della confutazione

Dimostrazione Risolutiva

Definizione: Sia F una formula proposizionale e G una formula in forma clausale equivalente a ¬F

- Una dimostrazione risolutiva proposizionale per F è una confutazione risolutiva per G.
- F è chiamata **teorema del calcolo risolutivo** se c'è una dimostrazione risolutiva per F e quindi se riusciamo a raggiungere un falso per G, ossia ¬F
- Denotiamo con $\vdash_r$ il fatto che **F ha una dimostrazione risolutiva**

Requisiti

- **Correttezza** se $\vdash_r F$ allora $\models F$ (se F ha una dim. risolutiva, allora F è valida)
- **Completezza** se $\models F$ allora $\vdash_r F$ (se F è valida, allora ha una dim. risolutiva)

Esempio

- Sia F una formula con n variabili proposizionali
- (F ^ (p ^ ¬p)) è insoddisfacibile
- Si dimostra in un singolo passo risolutivo ---> < [F], [p], [¬p] > dove [p], [¬p] ci da [ ]

Ulteriori metodi di risoluzione e Calcoli

Avendo degli oggetti di partenza molto semplici, tramite delle regole possiamo mettere insieme questi oggetti fino ad una formula finale, con una sola conclusione.

Definizione: Un calcolo logico consiste di:

- Un alfabeto (Nel nostro caso le lettere enunciative: p1, p2…)
- Un linguaggio (Le formule del calcolo degli enunciati)
- Un insieme di assiomi, ossia le formule alla base del nostro calcolo (questo insieme può essere finito o infinito, ma in realtà, dato che usiamo degli "schemi di assiomi" A → A può essere sostituito con infiniti valori).
- Un insieme di regole per mettere insieme gli oggetti ottenuti in precedenza

Inoltre

- Un calcolo si dice coerente se per ogni formula del linguaggio abbiamo che se F ha una dim. risolutiva ($\vdash_r F$) allora sappiamo F è valida ($\models F$)
- Un sistema di dice completo se F è valido ($\models F$) e ha una dim. risolutiva ($\vdash_r$)

Caratterizzazione dei Calcoli

- **positivo**, se i suoi assiomi sono validi
- **negativo**, se i suoi assiomi sono insoddisfacibili
- **generativo,** se parto dagli assiomi e cerco di ottenere la formula

- **analitico,** se partendo dalle conclusioni cerca di giungere agli assiomi

Deduzione Naturale

E' un modo per rappresentare ragionamenti logici con formalismi matematici

- Il suo alfabeto è lo stesso della logica proposizionale
- Ricorda che [ ] denota le formule insoddisfacibili
- Il suo linguaggio è lo stesso della logica proposizionale

- Suppose we want to show the validity of

$$((p \vee (q \wedge r)) \rightarrow ((p \vee q) \wedge (p \vee r))).$$

- A proof could look like the following:

Assume that $(p \vee (q \wedge r))$ holds. We distinguish two cases:
(i) Assume that p holds.
Hence, $(p \vee q)$ must hold.
(ii) Assume that $(q \wedge r)$ holds.
Hence, q and, consequently, $(p \vee q)$ must hold.
Thus, $(p \vee q)$ must hold in any case. (1)
Likewise, we conclude that $(p \vee r)$ must hold. (2)
From (1) and (2) we conclude that $((p \vee q) \wedge (p \vee r))$ holds.
We can now cancel the assumption and, thus, obtain
a proof of $((p \vee (q \wedge r)) \rightarrow ((p \vee q) \wedge (p \vee r)))$.

- Elimination of the implication

$$\frac{G \qquad (G \rightarrow F)}{F}(\rightarrow E)$$

- Schema which needs to be instantiated, e.g.,

$$\frac{(p \wedge q) \qquad ((p \wedge q) \rightarrow (q \wedge p))}{(q \wedge p)}(\rightarrow E)$$

Derivazioni: Notazione

- Le derivazione hanno forma di alberi dove i nodi sono etichettati da formule
- Le derivazioni sono denotate da ∇

- Una derivazione dove **la radice** è etichettata da **F** si denota con

∇
F

- Questa sarà chiamata derivazione di F
- **Una formula che etichetta un nodo foglia** in una derivazione è chiamata **ipotesi** o **assunzione** (in questo caso **G**)
- Se vogliamo individuare un'ipotesi G in una derivazione di F allora scriviamo

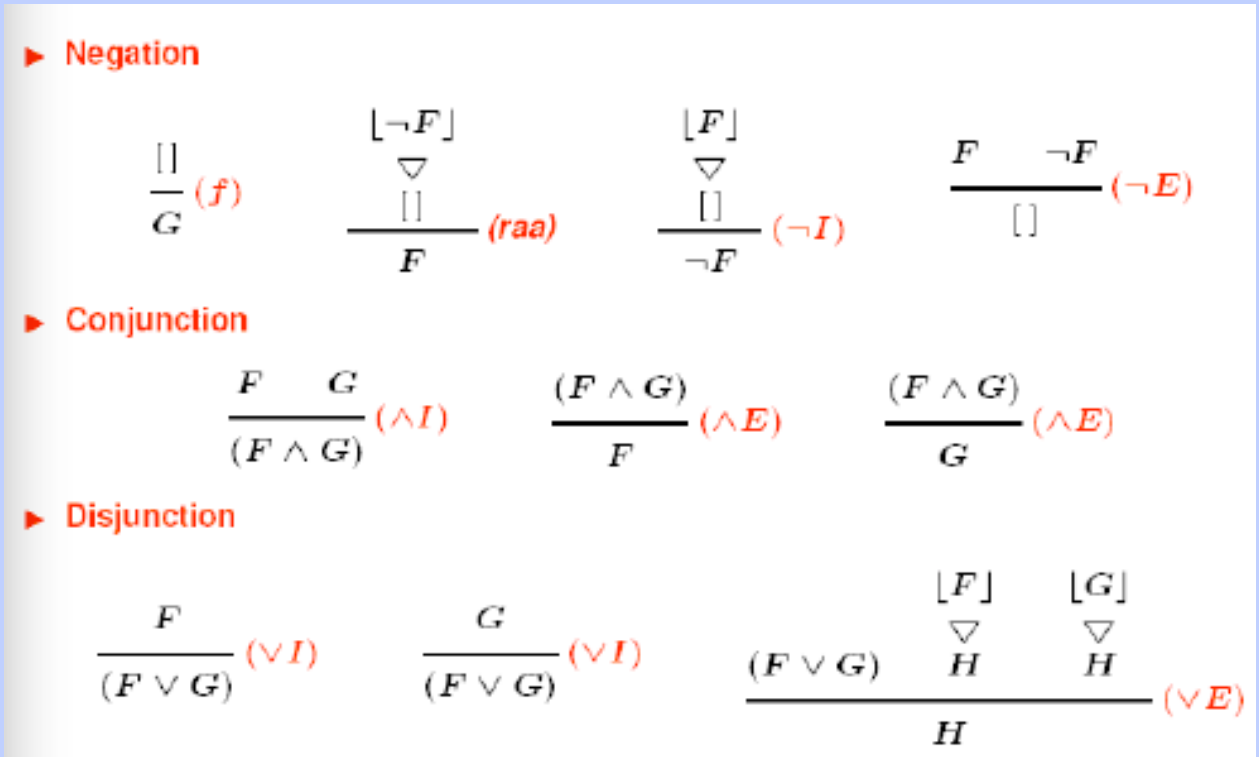
G
∇
F

Questa sarà chiamata **derivazione di F da G**

- Alcune regole di inferenza **annullano** le ipotesi
- Se un ipotesi è stata annullata, allora sarà denotata marcando l'ipotesi con [ ] (Si dice che *"scarica l'ipotesi"* . ******NB: il simbolo non è precisamente [ ], ma sono parentesi quadre senza il trattino orizzontale in alto**)
- Sia l'ipotesi annullata dall'applicazione della regola *a*, sia l'applicazione *a* stessa, saranno marcate con uno stesso indice univoco

Regole di Inferenza

Registrazione al tempo 1:10:24



- Implication

$$\dfrac{\begin{array}{c}[G]\\ \nabla\\ F\end{array}}{(G \to F)}\,(\to I) \qquad \dfrac{\begin{array}{c}[\neg F]\\ \nabla\\ \neg G\end{array}}{(G \to F)}\,(\to I) \qquad \dfrac{G \quad (G \to F)}{F}\,(\to E)$$

- Equivalence

$$\dfrac{\begin{array}{cc}[G] & [F]\\ \nabla & \nabla\\ F & G\end{array}}{(G \leftrightarrow F)}\,(\leftrightarrow I) \qquad \dfrac{G \quad (G \leftrightarrow F)}{F}\,(\leftrightarrow E) \qquad \dfrac{F \quad (G \leftrightarrow F)}{G}\,(\leftrightarrow E)$$

Derivazioni (1)

Definizione: l'insieme delle **derivazioni nel calcolo della deduzione naturale** è il più piccolo insieme X con le seguenti proprietà:

1. L (R) $\subseteq$ X
2. Se

$$\begin{array}{c}\nabla\\ H_1\end{array}$$

è una derivazione in X e:

$$\dfrac{H_1}{H_2}$$

è l'istanza di una regola r $\in$ { (f), (^E), (vI) }, allora

$$\dfrac{\begin{array}{c}\nabla\\ H_1\end{array}}{H_2}\,r$$

è una derivaizone in X

Derivazioni (2)

3. se:

$$\begin{array}{c} H_1 \\ \triangledown \\ H_2 \end{array}$$

è una derivazione in X, j è un nuovo indice e:

$$\dfrac{\begin{array}{c}[H_1] \\ \triangledown \\ H_2\end{array}}{H_3}$$

è un'istanza di una regola r ∈ { (raa), (¬I), (→ I) } allora:

$$\dfrac{\begin{array}{c}[H_1]^j \\ \triangledown \\ H_2\end{array}}{H_3}\ r^j$$

è una derivazione in X.
L'ipotesi H_1 è stata scaricata dall'applicazione della regola.

Derivazioni (3)

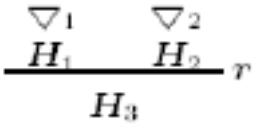
4. Se:

$$\begin{array}{cc} \triangledown_1 & \triangledown_2 \\ H_1 & H_2 \end{array}$$

sono derivazioni in X e se:

$$\dfrac{\begin{array}{cc} \triangledown_1 & \triangledown_2 \\ H_1 & H_2 \end{array}}{H_3}$$

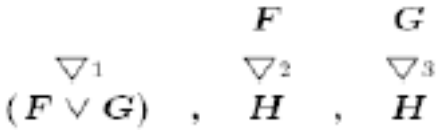
è un'istanza di una regola r ∈ { (¬E), (→ E) , (↔ E) } allora



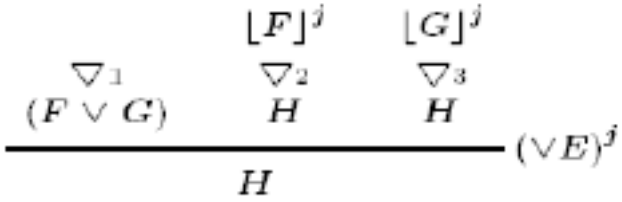
è una derivazione in X

Derivazioni (5)

5. Se:



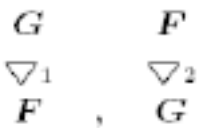
sono derivazioni in X e j è un nuovo indice, allora:



è una derivazione in X.
L'ipotesi F e G sono annullate dall'applicazione di (vE)

Derivazioni (6)

6. Se:



sono derivazioni in X e j è un nuovo indice allora:

$$\dfrac{\begin{array}{cc}[G]^j & [F]^j \\ \nabla_1 & \nabla_2 \\ F & G\end{array}}{(G \leftrightarrow F)}\ (\leftrightarrow I)^j$$

è una derivaizone in X.
Le ipotesi G e F sono annullate dall'applicazione di (↔ I)

Dimostrazione

- Un elemento in X è chiamato **derivazione di F nel calcolo della deduzione naturale** se nella sua rappresentazione ad albero il suo nodo radice è F
- Una **dimostrazione di F nel calcolo della deduzione naturale** è una derivazione di F nel calcolo della deduzione naturale, dove ogni ipotesi è scaricata. Ciò sarà denotato con $\vdash_n$ F.
- Un semplice esempio semplice è la dimostrazione di (p → p)

$$\dfrac{\dfrac{[\,p\,]^1}{}}{(p \rightarrow p)}\ (\rightarrow I)^1$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{[(p \vee (q \wedge r))]^1}\quad \dfrac{\dfrac{[p]^2}{(p \vee q)}(\vee I)\quad \dfrac{\dfrac{[(q \wedge r)]^2}{q}(\wedge E)}{(p \vee q)}(\vee I)}{(p \vee q)}(\vee E)^2 \quad \dfrac{\dfrac{\dfrac{}{[(p \vee (q \wedge r))]^1}\quad \dfrac{[p]^3}{(p \vee r)}(\vee I)\quad \dfrac{\dfrac{[(q \wedge r)]^3}{r}(\wedge E)}{(p \vee r)}(\vee I)}{(p \vee r)}(\vee E)^3}{((p \vee q) \wedge (p \vee r))}(\wedge I)}{((p \vee (q \wedge r)) \rightarrow ((p \vee q) \wedge (p \vee r)))}(\rightarrow I)^1$$

Calcolo dei Sequenti (Regole di Gentzen)

Definizione: un sequente è un **asserzione di derivabilità**, ossia un'espressione nella forma F ⊢ G (F segue G), dove F e G sono insiemi di formule.

Notazione: Si useranno le seguenti abbreviazioni:

```
{F1,…,Fn} → F1,…,Fn
F U {G} → F,G
```

Le regole di inferenza sono del tipo

```
      S1 … Sn
---------------- r
       S
```

Dove "r" identifica il nome della regola

oppure

```
--- r
 S
```

Assiomi, Taglio e Regole Strutturali

Axiom

$$\frac{}{H, \mathcal{F} \vdash \mathcal{G}, H}\ \text{ax}$$

Cut

$$\frac{\mathcal{F} \vdash \mathcal{G}, H \quad H, \mathcal{F} \vdash \mathcal{G}}{\mathcal{F} \vdash \mathcal{G}}\ \text{cut}$$

Structural rules

$$\frac{H, H, \mathcal{F} \vdash \mathcal{G}}{H, \mathcal{F} \vdash \mathcal{G}}\ \text{cl} \qquad \frac{\mathcal{F} \vdash \mathcal{G}, H, H}{\mathcal{F} \vdash \mathcal{G}, H}\ \text{cr}$$

Regole di Gentzen

Caso	Operazioni sul vero	Operazioni sul falso
∧	$\dfrac{\Gamma, a, b \vdash \Delta}{\Gamma, [\, a \wedge b \,] \vdash \Delta}$	$\dfrac{\Gamma \vdash a, \Delta \qquad \Gamma \vdash b, \Delta}{\Gamma \vdash [\, a \wedge b \,], \Delta}$
∨	$\dfrac{\Gamma, a \vdash \Delta \qquad \Gamma, b \vdash \Delta}{\Gamma\, [\, a \vee b \,] \vdash \Delta}$	$\dfrac{\Gamma \vdash a, b, \Delta}{\Gamma \vdash [\, a \vee b \,]\, \Delta}$
→	$\dfrac{\Gamma \vdash a, \Delta \qquad \Gamma, b \vdash \Delta}{\Gamma, [\, a \rightarrow b \,] \vdash \Delta}$	$\dfrac{\Gamma, a \vdash b, \Delta}{\Gamma \vdash [a \rightarrow b], \Delta}$
¬	$\dfrac{\Gamma \vdash a\, \Delta}{\Gamma, \neg a \vdash \Delta}$	$\dfrac{\Gamma\ , a \vdash \Delta}{\Gamma \vdash \neg a, \Delta}$

Coerenza e Compattezza

Teorema di Coerenza (Soundess)

Teorema: Una argomento è coerente (sound) se e solo se:

1. L'argomento è valido

2. Tutte le premesse sono vere

Per argomento si intende un insieme di una o più proposizioni, dette premesse e conclusioni.

Il teorema si può anche esprimere nei seguenti termini:

Sia F una proposizione, allora F è valida ($\models F$) se e solo se F ha una dimostrazione risolutiva ($\vdash_r F$)

Lemma sulla Risoluzione in Clause

Sia $F = <C_1,\ldots,C_n>$ una formula della logica proposizionale composta dalle clause C_i con $1 \leq i \leq n$, e siano $D_1,\ldots,D_m$ i risolventi calcolati in una derivazione risolutiva da F.

Allora $F \equiv <F,D1,\ldots,Dn>$

Applicazione del teorema di Coerenza per la risoluzione in Clause

- Supponiamo $\vdash_r F$
- Si arriva a questo tramite l'applicazione della risoluzione in clause.
- Dato che $\vdash_r F$ sappiamo che la dimostrazione risolutiva in clause contiene una clausa vuota [ ]
- Nel processo di risoluzione in clausa arriveremo ad avere $C_1, \ldots, C_2, [\,]$
- Grazie al Lemma al paragrafo precedente sappiamo che $\neg F \equiv <\neg F,C1,\ldots,Cn,[\,]>$
- Quindi $\neg F \equiv [\,]$, ossia $\neg F$ è insoddisfacibile
- Quindi F è valido $\models F$ (deriva da Validità e insoddisfacibilità, pag 11)

Applicazione del teorema di coerenza per i sequenti

Teorema da dimostrare: Una dimostrazione di un sequente S nel calcolo dei sequenti è definita così come segue:

- Un assioma nella forma seguente è una dimostrazione di S

$$\frac{}{S}\,r$$

- Se T1, …, Tn sono dimostrazioni di S1, … , Sn rispettivamente e il calcolo dei sequenti contiene una regola nella forma

$$\frac{S1 \quad \ldots \quad Sn}{S}\,r$$

allora ..

$$\frac{T1 \ \ldots \ Tn}{S}\ r$$

è la dimostrazione di S.

- Il calcolo proposizionale dei sequenti è coerente
- **Proprietà delle sottoformule:** tutte le formule che compaiono nelle condizioni di una regola, compaiono come sottoformule nelle conclusioni della regola stessa (quindi queste regole sono generative!).
  - Solo la regola del **taglio** viola questa proprietà
  - Genzen's "Hauptsatz": Il taglio può essere eliminato nelle dimostrazioni del calcolo dei sequenti.

Applicazione del teorema di Coerenza per le regole sui quantificatori di Gentzen

NB: L'argomento è trattato nel capitolo sulla logica dei predicati

Dimostrazione della coerenza della regola:

$$\frac{\Gamma,\ \forall x\ A(x),\ A(a_0)\ \vdash\ \Delta}{\Gamma,\ \forall x\ A(x)\ \vdash\ \Delta}$$

- Si supponga che la premessa F della regola

Introduzione al Teorema di Compattezza

Slide pag 80 I Registrazione del 5/11 alle 9.37

"Ho dei modelli di soddisfacibilità per degli insiemi finiti e voglio creare dei modelli di soddisfacibilità per insiemi infiniti"

Come possiamo definire soddisfacibile un insieme infinito di formule?

Idea: testiamo tutti i sottoinsiemi finiti!

Ricorda che $\equiv$ è una relazione di equivalenza su L(R)

Sia L(R,n) $\subseteq$ L(R) in cui sono presenti solo le variabili p1, …, pn. Ci sono 2^n differenti interpretazioni per L(R,n) e saranno definite $2^{2\wedge n}$ classi di equivalenza su L(R,n)

Teorema di Compattezza

pag 73

Teorema: Dato un insieme F di formule proposizionali che sono soddisfacibili, per ogni insieme finito G incluso in F, anche G è soddisfacibile.

Dimostrazione: Sia F un insieme di formule possibilmente infinito. Mostriamo che:

(a) Se F è soddisfacibile, allora lo è ogni sottoinsieme G $\subseteq$ F

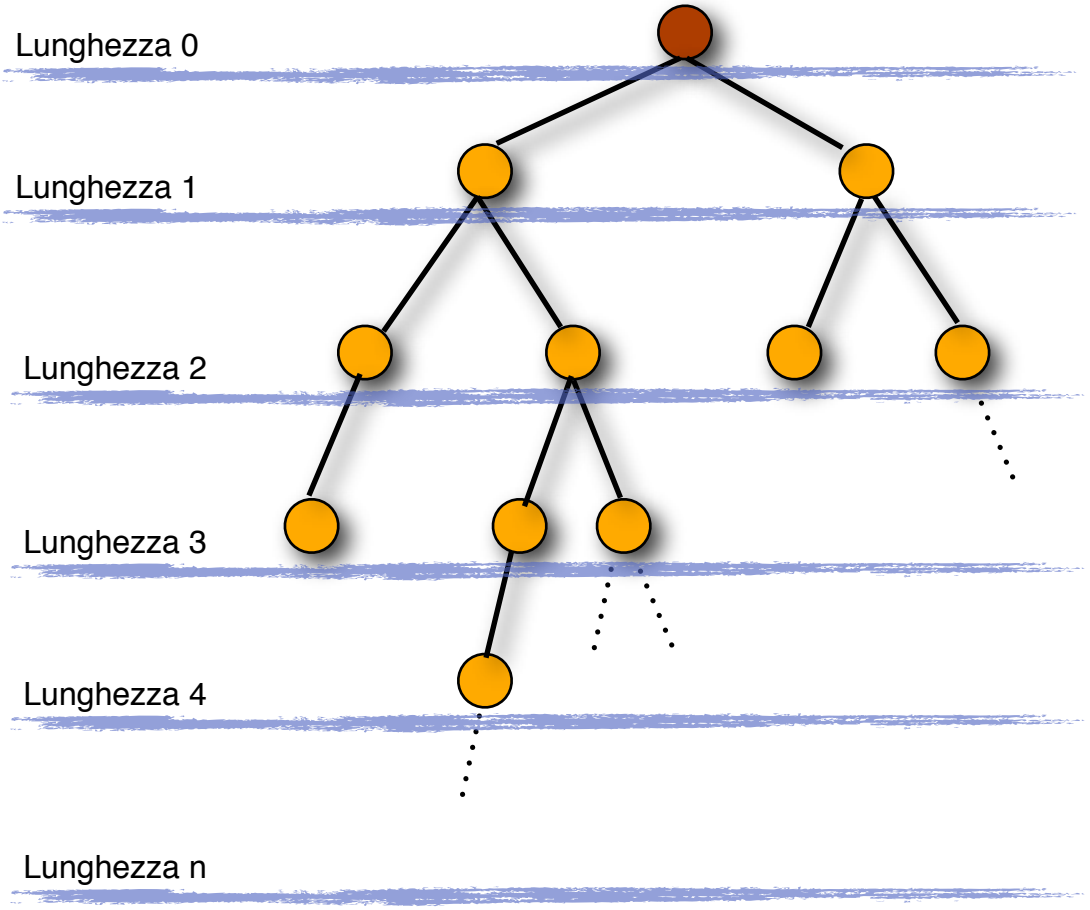
(b) Se ogni insieme G $\subseteq$ F è soddisfacibile, allora lo è anche F

a. Segue immediatamente dalla definizione di soddisfacibilità per un insieme

b. Dobbiamo dimostrare l'esistenza di un modello per F a partire dai modelli di ogni insieme finito G ⊆ F. *(Pensa ad un insieme infinito qualsiasi. So che per tutti i suoi sottoinsiemi vale una proprietà e voglio dimostrare che quella proprietà vale anche per quello infinito)*

Dimostrazione del teorema di compattezza (della parte b)

Per la dimostrazione del teorema di compattezza faremo uso degli alberi binari.



Nell'albero rappresentato possiamo trovare cammini di lunghezza 0, 1, 2, 3, … Questo albero binario è infinito. In sostanza questo albero è costruito in modo da poter trovare cammini di lunghezza n con n $\in \mathbb{N}$. Se ciò è vero, allora posso trovare un **cammino infinito**.

Premesse:

- Preso un nodo qualsiasi in cui passano cammini di lunghezza arbitraria
- Ogni nodo figlio può avere come figli o un solo nodo o due nodi.

  (a) Se ha un nodo solo allora vuol dire che esiste anche per quel nodo un albero di lunghezza arbitraria

  (b) Se ha due nodi a1 e a2 e non vi sono cammini di lunghezza n1 e n2 (due numeri arbitrari) allora non esiste neanche un cammino di lunghezza arbitraria al padre di questi a1, a2

Dimostrazione: Voglio provare che preso un nodo con cammini di lunghezza arbitraria, questa proprietà vale anche per i suoi figli.

- Prendo un nodo qualsiasi da cui posso costruire cammini arbitrariamente lunghi

  (a) Se ha 1 solo nodo figlio per cui non vale la stessa proprietà (avere cammini arbitrari), allora la proprietà non varrebbe neppure per il padre!

  (b) Se un nodo ha 2 figli allora dobbiamo considerare i cammini passanti da a1 e quelli passanti da a2:
  - a1: cammini di lunghezza n1
  - a2: cammini di lunghezza n2
  - Prendiamo ora il massimo tra n1 ed n2.
  - Se non passano cammini di lunghezza n1 da a1 e di lunghezza n2 da a2, allora non passano cammini arbitrariamente lunghi di lunghezza MAX(n1,n2) + 1 per il nodo padre!

Ora vediamo la situazione al contrario:

- Prendo un nodo a da cui passano cammini arbitrariamente lunghi. Faccio “scendere” la proprietà sui suoi nodi.
- Se ho 1 solo figlio su cui non vale la proprietà dei cammini arbitrariamente lunghi, allora non vale neppure sul padre, il quale non avrà n+1 cammini arbitrariamente lunghi
- Se ho 2 figli la proprietà può valere per l’uno o per l’altro. Prendiamo il MAX(n1, n2), ossia il massimo numero di cammini tra quelli dei nodi figli. Se non vale la proprietà per nessuno dei due figli allora vuol dire che non partono cammini arbitrariamente lunghi dal padre di lunghezza MAX(n1,n2) + 1.

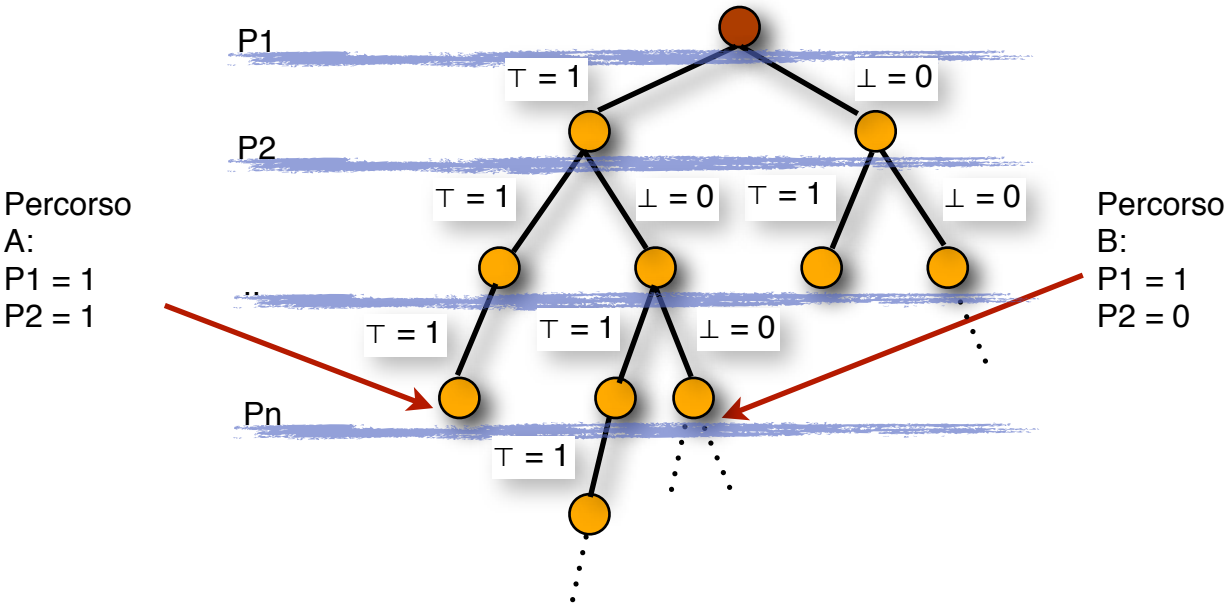
Quindi se vale la proprietà dei cammini arbitrariamente lunghi per la radice allora vale per uno dei due figli.

Nota che io so che vale per uno dei due figli, ma non so per quale dei due! (questa dimostrazione non è quindi costruttiva).

Teorema di compattezza (..segue)

Minuto 17

Ho dei teoremi di soddisfacibilità per sottoinsiemi finiti e devo costruire un modello di soddisfacibilità per un insieme infinito.



- Costruisco un albero binario molto grande e ad ogni livello faccio corrispondere una variabile enunciativa
-
- Ora il teorema di compattezza dice: *dati tutti i modelli dei sottoinsiemi finiti, riesco a costruire un modello dell'insieme infinito*
- Prendiamo il sottoinsieme finito che ha una sola variabile enunciativa. Quindi P1.
- P1 avrà valore 0 oppure 1 per far sì che il modello sia vero.
- Poi prendiamo un altro sottoinsieme finito che ha al massimo 2 variabili enunciative. Quindi P1 e P2.
- Di nuovo, per far sì che il modello sia vero, P1 e P2 dovranno avere due particolari valori di verità **(E' proprio così?)**
- Dopodiché prendiamo un sottoinsieme finito con 3 variabili enunciative: P1, P2 e P3.
- Per dare un valore di verità a queste tre variabili dobbiamo avere un cammino in questo albero che ci da un valore vero alla fonte (?)

- Come faccio a dire che, se esiste un percorso di lunghezza n, allora so automaticamente che esiste un percorso di lunghezza n+1??? Non è ammesso che un nodo non abbia figli? O.o
- Che vuol dire che un percorso è un modello per un insieme di formule?
- Quale percorso nell'albero è un modello??

Logica dei Predicati

Slide “Predicati” I Registrazione al minuto 59 e(continua la precedente)

Sintassi

registrazione del 06/11/09 alle 9.09

L’alfabeto della logica dei predicati o **logica del prim’ordine** (in cui quantifichiamo solo e soltanto sulle variabili) consiste in:

- **Predicati**: Un insieme R finito o potenzialmente infinito (che ha al max la cardinalità di $\mathbb{N}$) di simboli relazionali
- Un insieme L finito o potenzialmente infinito di **simboli funzionali (o funzioni)**
- Un insieme V finito o potenzialmente infinito di **variabili**
- **Connettivi**: {¬/1, ^/2, v/2, .. } (i soliti..) più i **caratteri speciali** { (,), , }
- $\forall$ è chiamato **quantificatore universale**
- $\exists$ è chiamato **quantificatore esistenziale**.
- Arity n $\in \mathbb{N}$ (L’**arità** di n) è assegnato ad ogni funzione. E’ il numero di parametri richiesti da quella funzione. (Es: la somma ha arità 2 perchè ha 2 parametri: x + y. L’elevamento a potenza ha alirtà 1 perchè ha 1 parametro: x^2). NB: le costanti sono simboli funzionali con 0 posti.

Notazione

Indichiamo con R, F e V i seguenti insiemi (dati così come sono e non discussi):

- R: Relazioni
- F: Funzioni
- V: Variabili

Inoltre abbiamo:

- l’insieme T(F,V) dei termini.

Notation		
	$p, q, \ldots$	relation symbols,
	p/n	relation symbol with arity n,
	$g, h, \ldots$	function symbols,
	g/n	function symbol with arity n,
	$a, b, \ldots$	constant symbols,
	$X, Y, \ldots$	variables,

Termini

Definizione: L'insieme T(F,V) di termini* è il più piccolo insieme che soddisfa le seguenti condizioni:

1. Ogni variabile X ∈ V è un termine
2. se g/n ∈ F allora {t1,...tn} ⊆ T (F,V), allora la stringa g(t1,…,tn) è un termine**

*NB: T sta per l'insieme di termini, F per le funzioni e V per le variabili.

**NB: Si scrive g anzichè g() per comodità

In pratica i termini sono variabili proposizionali oppure se G è un simbolo funzionale e T1 e Tn sono termini allora G(T1, ..., Tn) è un termine.

Formule della logica dei predicati

Definizione: Sono formule atomiche nella forma p(t1,……,tn) dove p/n è un predicato con arità n e {t1, …. , tn) è un insieme di termini.

L'insieme di formule dei predicati è il più piccolo insieme che rispetta le seguenti caratteristiche:

1. Ogni atomo è una formula
2. Se F è una formula, allora lo è anche ¬F
3. Se F1 e F2 sono formule e •/2 è un connettivo binario allora (F1 • F2) è una formula
4. Se F è una formula, Q un quantificatore e X è una variabile, allora (QX)F è ancora una formula.*

*NB: X può anche non apparire in F!!!

C'è un esempio fatto alla lavagna che forse ha Sergio

Induzione Strutturale

Voglio introdurre un'induzione strutturale sugli oggetti definiti da questo linguaggio.

Prendiamo per esempio i termini che sono definiti in base alle Variabili e ai Funtori e dato un Funtore di arità n posso mettere n termini su questo funtore.

Se volessimo provare una proprietà E sull'insieme T(F,V) dei termini, dobbiamo prima provare che è valida per la base, ossia le variabili, poi effettuare il passo induttivo cioè: se è vero per gli oggetti che "stanno dentro" (le variabili) allora è anche vero per l'oggetto che si ottiene applicando la funzione F sull'oggetto per cui è già vera la proprietà E.

Ricorsione strutturale

Funziona come per il calcolo enunciativo: un programma che gestisce tutte le formule del calcolo enunciativo. Questo ha bisogno di lavorare prima sulle formule più semplici per poi passare su quelle più complicate formate da quelle più semplici.

- Troviamo prima la base, in cui il nostro programma lavora sulle variabili e sui simboli costanti
- Passo Ricorsivo: dagli elementi della base, costruisce elementi più complessi.

In pratica: abbiamo la capacità di lavorare su t1,.., tn, dobbiamo provare di poter lavorare anche su f(t1,…,tn)

Esempio:

$$foo(T) = \begin{cases} 0 & \text{if } T \text{ is a variable,} \\ 1 & \text{if } T \text{ is a constant symbol,} \\ \sum_{i=1}^{n} foo(t_i) & \text{if } T \text{ is of the form } f(t_1, \ldots, t_n). \end{cases}$$

Sottotermini e sottoformule

E' un ampliamento del metodo per trovare le sottoforumule nel calcolo degli enunciati.
Esempio:

$\{g(f(X), Y),\ f(X),\ X,\ Y\}$
is the set of subterms of the term $g(f(X), Y)$.

$\{\ (\forall X)(\exists Y)\,(q(X) \rightarrow p(g(a,b), f(f(Y)))),$
$(\exists Y)\,(q(X) \rightarrow p(g(a,b), f(f(Y)))),$
$(q(X) \rightarrow p(g(a,b), f(f(Y)))),$
$q(X),$
$p(g(a,b), f(f(Y)))\ \}$
is the set of subformulas of the formula
$(\forall X)(\exists Y)\,(q(X) \rightarrow p(g(a,b), f(f(Y)))).$

Variabili libere e limitate

- **Variabiil libere**: quelle non limitate da un qualificatore. Oggetti a(x,y).
  - In una formula atomica, le variabili che compaiono sono tutte libere.
  - Se ho ¬F, le variabili libere sono tutte quelle di F.
  - Se ho invece F1 • F2, le variabili libere sono tutte le variabili libere di F1 e F2
  - Le variabili libere nella formula (QX)F sono tutte le variabili libere in F escluso X
- **Variabili limitate**: quelle limitate da un qualificatore

registrazione alle 9.29

Esempio alla lavagna 5

Termini chiusi e formule

Un termine è chiuso se non contiene alcuna occorrenza di una variabile.

Definizione: Una formula chiusa è una formula dove ogni variabile è limitata. Le formule chiuse possono essere solo “vero” o “falso”.

Sostituzioni

registraizone alle 9.37

Definizione: Una sostituzione è una funzione che va da σ: V → T (F,V), dove solo un numero finito di variabili non è mappata su se stessa.

$$dom(\sigma) = \{X \mid X \in \mathcal{V} \text{ and } \sigma(X) \neq X\}.$$
$|dom(\sigma)|$ is finite.
σ can be represented by a finite set of pairs

$$\{X \mapsto \sigma(X) \mid X \in dom(\sigma)\}$$

dove X è la variabile da sostituire e σ(X) è la sua sostituzione

If $dom(\sigma) = \emptyset$, then σ is called empty substitution.
ε denotes an empty substitution.
The restriction of σ to a set $\mathcal{U} \subseteq \mathcal{V}$ of variables is defined as

$$\sigma|_{\mathcal{U}} = \{X \mapsto t \mid X \mapsto t \in \sigma \text{ and } X \in \mathcal{U}\}.$$

Un esempio di sostituzione che permette di sostituire tutte le occorrenze di x+1 con x

$$\forall\lambda \ \exists y \ (x+1=y) \ \rightarrow \ \forall\lambda \ \exists x \ (x+1=x)$$

Applicazione delle sostituzioni

Come costruire un processo automatico di sostituzione?

NB: D’ora in poi scrivo Xσ anziché σ(X).

Definizione: sia σ una sostituzione, σ : V → T (F,V) è un’estensione a σ* : T(F,V) → T(F,V) come segue:

Sia t ∈ T(F,V), allora: (nota che σ* = $\hat{\sigma}$)

$$t\hat{\sigma} = \begin{cases} t\sigma & \text{if } t \in \mathcal{V}, \\ t & \text{if } t \text{ is a constant}, \\ f(t_1\hat{\sigma}, \ldots, t_n\hat{\sigma}) & \text{if } t \text{ is of the form } f(t_1, \ldots, t_n). \end{cases}$$

Esempio:

$$t=(t_1 \bullet t_2) \rightarrow t\sigma=(t_1\sigma \bullet t_2\sigma)$$
$$t=\neg t_1 \rightarrow t\sigma=\neg(t_1\sigma)$$

- tσ* è chiamata **istanza** di t sotto σ*
- se tσ* è chiusa, allora tσ* è chiamata **ground instance** di t sotto σ* e σ* è chiamata **ground substitution** per t.

Composizione di sostituzioni

Definizione: siano σ e θ sostituzioni. La **composizione** σθ di σ e θ è la sostituzione definita come segue:

$$\{X \mapsto t\hat{\theta} \mid X \mapsto t \in \sigma \text{ and } X \neq t\hat{\theta}\}$$
$$\cup \{Y \mapsto s \mid Y \mapsto s \in \theta \text{ and } Y \notin dom(\sigma)\}$$

- In pratica devo applicare prima la sostituzione σ a X, ottenendo t; poi devo applicare a t la sostituzione θ. La sostituzione può essere fatta solo quando da t non torno a X tramite θ, altrimenti sarebbe inutile.
- Se non posso applicare la prima sostituzione, allora applico la seconda (?)

Varianti

Definizione: Siano E1 ed E2 due termini (formule) della logica predicativa, questi si dicono Varianti se vi sono delle sostituzioni che ci permettono di partire da E1 e costruire E2 (o viceversa).

I varianti sono due formule che, a meno di sostituzioni, sono equivalenti. Quindi **posso passare da una formula all'altra tramite sostituzioni**.

Due formule che sono varianti tra loro, **hanno lo stesso significato semantico**.

Semantica

Vedi slide 18

Sia D un'insieme:

$$\begin{array}{lcll} D^2 & = & D \times D & = & \{(d_1, d_2) \mid d_1 \in D \text{ and } d_2 \in D\} \\ D^n & = & D^{n-1} \times D & = & \{(t, d_n) \mid t \in D^{n-1} \text{ and } d_n \in D\} \\ & & & =: & \{(d_1, \ldots, d_n) \mid d_i \in D,\ 1 \leq i \leq n\},\ n > 2 \\ D^1 & = & D & =: & \{(d) \mid d \in D\} \\ D^0 & = & & = & \{()\} \end{array}$$

Per relazioni si intende:

$$\begin{array}{ll} R \subseteq D^2 & R = \{(n, m) \mid n, m \in \mathbb{N} \text{ and } n < m\} \\ R \subseteq D^3 & R = \{(x, y, z) \mid x, y, z \in \mathbb{N} \text{ and } x^2 + y^2 = z^2\} \\ R \subseteq D^1 & R = \{(n) \mid n \in \mathbb{N} \text{ and } n \text{ even}\} \\ R \subseteq D^0 & R = \emptyset \text{ or } R = \{()\} \end{array}$$

Interpretazioni e Modelli

Vogliamo dare un'interpretazione al nostro linguaggio L(R,F,V).

Partendo da un insieme D non vuoto, ad ogni funzione g/n ∈ F associamo una funzione ennaria nel modello chiamata g^I per ricordare che è un'interpretazione del simbolo formale g del linguaggio L(R,F,V), dove $g^I: D^n \rightarrow D$.

Una relazione ennaria R sarà interpretata come una relazione p^I contenuta in D^n.

Non usiamo vero e falso, semplicemente includiamo nell'insieme solo tutti quelli veri.

D è chiamato **dominio** dell'interpretazione.

Interpretazione delle Costanti

Le costanti sono interpretate ...come costanti!

Assegnazione di Variabili

Definizione: data un'interpretazione $I = (D, \cdot^I)$, un'**assegnazione di variabile** è una funzione (di interpretazione) $Z : V \rightarrow D$ che associa un elemento del dominio D ad una variabile.
Quindi, data X, la sua **immagine** nel dominio D è denotata da X^Z, se Z è la funzione di interpretazione. Ciò significa che la funzione Z assocerà alla variabile X un elemento del dominio D.
Nota che alla stessa variabile posso associare diverse interpretazioni. Potrei associare Z1 che alla stessa variabile associa un altro elemento del dominio.

Sapendo che V è l'insieme delle variabili $a_1, \ldots, a_n$, sia Y ∈ V.

Sia Z l'assegnazione di variabile.

Sia d ∈ D.

Scrivere Y^Z, significa che Y è una variabile che può essere uno qualsiasi tra {a1,…,an}.

Scrivere $Y^{\{X \to d\}Z}$ significa che se Y è uguale a X allora scrivo d, altrimenti scrivo la vecchia interpretazione:

$$Y^{\{X \to d\}Z} = \begin{cases} d & \text{if } Y = X, \\ Y^Z & \text{otherwise.} \end{cases}$$

Interpretazione dei Termini

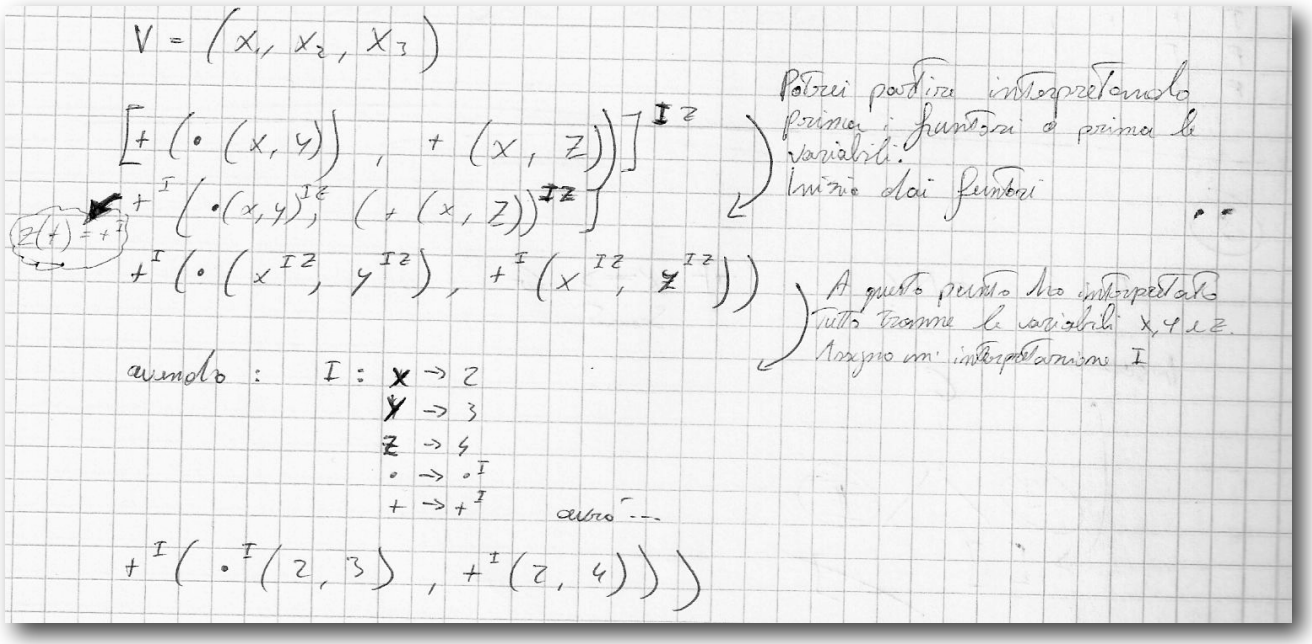
Definizione: Data un'interpretazione I = (D,•^{I}) e un'assegnazione di variabile Z rispetto ad I, chiamiamo **interpretazione del termine** t ∈ T(F,V) il simbolismo $t^{I,Z}$. Questo è definito come segue:

1. Per ogni variabile X ∈ V, abbiamo che $X^{I,Z} = X^Z$
2. Per ogni termine nella forma complessa g(t1,…,t2) abbiamo:

$$[g(t_1,\ldots,t_n)]^{I,Z} = g^I(t_1^{I,Z},\ldots,t_n^{I,Z})$$

dove g/n ∈ F e $t_1,\ldots,t_n$ sono termini.

Ascolta registrazione al minuto 24.40 (Nota che inizialmente non usa le variabili x,y,z)



Interpretazione delle Formule (inizia..)

L'interpretazione di una funzione mi restituisce un oggetto, mentre l'interpretazione di una formula mi da o vero o falso.

Definizione: Data un'interpretazione I = (D,•^{I}) e un'assegnazione di variabile Z rispetto ad I, I e Z assegnano ad ogni formula F ∈ L(R,F,V) un valore di verità $F^{I,Z}$ così come segue:

1. $[p(t_1, \ldots, t_n)]^{I,Z} = \top$ iff $(t_1^{I,Z}, \ldots, t_n^{I,Z}) \in p^I$.
2. $[\neg F]^{I,Z} = \neg^*(F^{I,Z})$.
3. $[(F_1 \circ F_2)]^{I,Z} = (F_1^{I,Z} \circ^* F_2^{I,Z})$ for all binary connectives $\circ/2$.
4. $[(\forall X)\, F]^{I,Z} = \top$ iff for all $d \in \mathcal{D}$: $F^{I,\{X \rightarrow d\}Z} = \top$.
5. $[(\exists X)\, F]^{I,Z} = \top$ iff for some $d \in \mathcal{D}$: $F^{I,\{X \rightarrow d\}Z} = \top$.

Nel punto 4 e 5 devo fare attenzione in quanto la presenza dei quantificatori deve far cambiare il metodo in cui eseguo l'interpretazione: in questo caso la X varia su tutti gli elementi del dominio, non più su uno solo, e la condizione che si cerca deve valere su tutti (nel caso del ∀ o su alcuni nel caso del ∃)

Ricapitolazione

Ascolta ricapitolazione al minuto 39

Come è fatto il nostro linguaggio?

- **Costanti**: rimangono costanti.
- **Variabili**: vengono interpretate come elementi del dominio D.
- **Funzioni** (**funtori**): la funzione g/n (funzione g con arità n) viene interpretata come una funzione che va da $D^n \rightarrow D$ (dove D è il dominio). Se una funzione al suo interno ha altre funzioni, le devo interpretare tutte ricorsivamente.
- **Formule**: Si interpretano come sempre. L'unica novità è la presenza dei simboli ∀ e ∃. Data l'intepretazione di partenza, in presenza di un ∃x o ∀x (per esempio), devo effettuare l'interpretazione di partenza, cambiandola per la x. Questa x diventa una variabile che assume tutti i possibili valori in D. Nel caso del ∀ devo avere sempre "vero", nel caso del ∃ mi basta che almeno un'interpretazione sia "vero".

Interpretazione di Formule (..continua)

registrazione 12/11/09 alle 09.09

Sia D il nostro dominio di interpretazione delle nostre variabili.

L'interpretazione è definita come I = (D, $\bullet^I$)

Esempio:

- p (X1, X2)
- Definisco il dominio per X1 e X2: D = $\mathbb{N}$.
- Ciò vuol dire che ad X1 e X2 posso associare un qualsiasi n $\in \mathbb{N}$
- il predicato P sarà: P : $\mathbb{N}$ x $\mathbb{N}$ → { V, F}
- Quindi ho Coppie di (x, y) p (x,y) ≡ V

Oppure posso dire

- x,y ∈ D se e solo se p(x,y) = V *(La V indica Vero)*

Predicato: è un programma che andrà interpretato

Termine: una funzione che dato un oggetto del dominio, mi da un oggetto del Codominio.

Quando scrivo $T_1^{I,Z}$ voglio dire: prendo le interpretazioni degli oggetti da I e le variabili da Z

NB: La semantica è la tabella con tutti i valori di vero o falso delle nostre funzioni!

Sostituzioni e assegnazioni di variabili

inizia dal minuto 25

Se da un termine T tramite **un'interpretazione** I e **un'assegnazione di variabile** Z ottengo d, allora posso andare a sostituire in tutto il predicato T con d.

Esempio:

- P (succ(x), +(x2, x1))
- Se assegno a +(x2, x1) il valore 6, allora posso scrivere questo valore al posto del termine:
- P (succ(x), 6)

1. **$[s\{Y \rightarrow t\}]^{I,Z} = [s]^{I,\{Y \rightarrow d\}Z}$**
   Esempio:
   + (5, + (3, 2))
   - ho l'interpretazione che mi dice +(3,2) = 5 quindi sostituisco +(3,2) con la sua interpretazione.
   - allora ho +(5,5)
2. **$[G\{Y \rightarrow t\}]^{I,Z} = [G]^{I,\{Y \rightarrow d\}Z}$ se è libero per Y in G**
   Anziché sostituire una variabile con la sua interpretazione, **posso sostituire una variabile con un altro oggetto, purché questo oggetto sia libero***!*
   Ovviamente il nuovo oggetto deve essere equivalente al precedente
   Esempio :
   se ho + (5, + (x,**y**)) posso anche scrivere + (5, +(x,**z**))
   se ho + (5, + (x,**y**)) NON posso scrivere + (5, +(x,**x**))!!

La stessa cosa scritta nell'esempio al punto 2 posso farla per le formule:

- ∀x esiste y P(x,y) dove la loro interpretazione è: x,y ∈ ℕ e P < (ossia P è la funzione "minore di")
- x ≡ y
- Sostituisco x ad y, commettendo volontariamente un errore in quanto x non è libera.
- ∀x esiste x P(x,x) quindi x<x (assurdo!!!)

- Avrei potuto invece sostituire la y con z, ma non con x o un'altra variabile non libera.

Modelli

Definizione: Si dice che I è modello per F e si indica con $I \models F$ se e solo se $F^I = \top$

Prendiamo due formule:

- $[\ \forall x\ \exists y\ P(x,y)\ ]^{I,z}$
  qui non ho alcuna variabile perché sia x che y sono quantificate dal $\forall$ e da $\exists$.
  Quindi in questo caso Z (che significa "assegna alle variabili libere i valori ..") non ha significato, perché non influisce con la formula.
- $[\ \exists y\ \ P(x,y)\ ]^{I,Z}$
  Qui il risultato dipende dal valore di x, che è libera. In questo caso, avendo P interpretato come "minore di" esiste sempre un'interpretazione di quella formula che mi da "vero", ma in generale ciò non è detto.
  Per esempio assegnamo x = 5.
  In questo caso la formula è vera, ma se avessi assegnato un altro valore ciò poteva non essere. Quindi il valore di verità di F dipende anche dal valore (dalla z) che io assegno alle variabili.

A differenza del calcolo degli enunciati, quindi, se ho una formula in cui sono presenti della variabili libere, la stessa formula può darmi valore vero o falso a seconda dell'interpretazione che ne do (a seconda della Z). Invece se tutte le variabili fossero quantificate la formula mi darebbe soltanto o vero o falso indistintamente dall'interpretazione.

Il punto è cercare di **fare in modo che** il calcolo dei predicati si semplifichi come nel calcolo degli enunciati in cui **una formula può essere o vera o falsa** (anche se non sempre è possibile).

Some remarks

D'ora in poi si cercherà di rendere le formule chiuse.

Molte nozioni della logica degli enunciati, possono essere estese alla logica dei predicati:

- Validità, sodisfacibilità, falsicabilità, insoddisfacibilità
- Una formula chiusa F si dice valida se e solo se tutte le sue interpretazioni sono modelli per F
- Una formula chiusa F è valida se e solo se ¬F è insoddisfacibile
- Siano F,F1,…,Fn formule chiuse.

$$\{F1,\ldots,Fn\} \models F \text{ se e solo se } (\langle F1,\ldots,Fn\rangle \rightarrow F)$$

Conseguenza Logica

Definizione: Una formula chiusa F è una conseguenza dell'insieme G di formule chiuse (in simboli $G \models F$) se e solo se ogni modello per G è anche un modello per F

Proposizionale vs. Logica del primo ordine

- Se tutte le relazioni sono nullarie, ossia sono soltanto delle costanti, allora la logica del prim'ordine è uguale a quella proposizionale
- Se tutte le variabili sono quantificate, allora la logica del prim'ordine è equivalente alla logica proposizionale

Interpretazione all'Herbrand

Registrazione al minuto 66.40

- Assumiamo che il nostro dominio è formato da 2 oggetti {a,b}
- Se io ho una variabile, so che potrà essere uguale ad *a* o a *b*
- Ho una funzione F(x).
- Dico che se ho F(a), allora ha interpretazione F(b) e viceversa.
- Dico poi che FF(a) ha interpretazione FF(b) e viceversa.
- e così via...

Quindi si parte dando un'interpretazione agli oggetti costanti iniziali, poi se abbiamo dei funtori diamo un'interpretazione ai funtori sugli oggetti di partenza. L'interpretazione finale sarà proprio la stringa che deriva dall'interpretazione di tutti gli oggetti della formula (FF(a) F(b)aF(a)…)

D'ora in poi assumiamo sempre che il nostro insieme delle formule contiene almeno una costante. Nel caso in cui non c'è, la inseriamo noi.

1. Abbiamo un dominio D = T (F) che è chiamato Universo di Herbrand
2. Per ogni t ∈ T (F) abbiamo che l'interpretazione t^I = t (Quindi un l'interpretazione di un termine è il termine stesso!)
   Esempio: abbiamo la formula:

   `F = {a/0, b/0, h/1, g/2}`

   a e b sono chiaramente due costanti in quanto hanno arità 0, mentre g e h sono due funtori.
   Il cui universo di herbrand è:

`T(F) ≡ {a, b, h(a), h(b), g(a,a), g(a,b), g(b,b), h(h(a)),…}`

Interpretazione all'Herbrand e formule

Consideriamo il linguaggio:

`L({p/1,q/1,r/1},{g/1,a/0},V)`

quindi:

- 3 lettere predicative ad 1 posto
- 1 funtore ad 1 posto ed 1 costante

- simboli variabili

e la formula:

$$F = ((\forall X)(p(X) \vee q(X)) \wedge (\forall Y) r(g(Y)))$$

il cui dominio all'Herbrand è:

$$T(F) = \{a, g(a), g(a), g(g(a)), \ldots\}$$

L'interpretazione all'Herbrand deve essere induttiva, deve partire dalle costanti e deve essere chiusa rispetto a tutti i funtori. Quindi l'interpretazione di ogni predicato deve corrispondere col dominio all'Herbrand, ossia **deve racchiudere tutte le possibili combinazioni tra le costanti e i funtori, e anche tra i funtori e i funtori stessi**.

Esempio (solo I^1 è un'interpretazione all'Herbrand):

I_1: $p^{I_1} = q^{I_1} = r^{I_1} = \mathcal{T}(\mathcal{F})$. $\rightsquigarrow$ I_1 is a model for F.

I_2: $p^{I_2} = q^{I_2} = r^{I_2} = \emptyset$. $\rightsquigarrow$ I_2 is not a model for F.

I_3: $p^{I_3} = \{g(g(a)), g(g(g(g(a)))), \ldots\}, q^{I_3} = \{a, g(a)\}, r^{I_3} = \{g(g(a))\}$.
$\rightsquigarrow$ I_3 is not a model for F.

Equivalenze e forme normali

Equivalenza semantica

In questa sezione le formule potrebbero anche non essere chiuse.

- Due formule si dicono semanticamente equivalenti e si scrive F ≡ G se per tutte le interpretazioni I e tutte le assegnazioni di variabili Z avremo che $F^{I,Z} = G^{I,Z}$.
- Nota che ora abbiamo la Z oltre la I in quanto le formule non sono chiuse, quindi dobbiamo assegnare dei valori alle variabili libere
- Tutte le regole di equivalenza semantica per la logica proposizionale, valgono anche per le formule del prim'ordine

$(F \wedge F)$	$\equiv$	F	
$(F \vee F)$	$\equiv$	F	idempotency
$(F \wedge G)$	$\equiv$	$(G \wedge F)$	
$(F \vee G)$	$\equiv$	$(G \vee F)$	commutativity
$((F \wedge G) \wedge H)$	$\equiv$	$(F \wedge (G \wedge H))$	
$((F \vee G) \vee H)$	$\equiv$	$(F \vee (G \vee H))$	associativity
$((F \wedge G) \vee F)$	$\equiv$	F	
$((F \vee G) \wedge F)$	$\equiv$	F	absorption
$(F \wedge (G \vee H))$	$\equiv$	$((F \wedge G) \vee (F \wedge H))$	
$(F \vee (G \wedge H))$	$\equiv$	$((F \vee G) \wedge (F \vee H))$	distributivity

$\neg\neg F$	$\equiv$	F	double negation
$\neg(F \wedge G)$	$\equiv$	$(\neg F \vee \neg G)$	
$\neg(F \vee G)$	$\equiv$	$(\neg F \wedge \neg G)$	de Morgan
$(F \vee G)$	$\equiv$	F, if F is valid	
$(F \wedge G)$	$\equiv$	G, if F is valid	tautology
$(F \vee G)$	$\equiv$	G, if F is unsatisfiable	
$(F \wedge G)$	$\equiv$	F, if F is unsatisfiable	unsatisfiability
$(F \leftrightarrow G)$	$\equiv$	$((F \wedge G) \vee (\neg G \wedge \neg F))$	equivalence
$(F \rightarrow G)$	$\equiv$	$(\neg F \vee G)$	implication

- In più ci sono:

Theorem 4.25 Let F and G be formulas. The following equivalences hold:

$$\begin{aligned}
\neg(\forall X)\, F &\equiv (\exists X)\, \neg F \\
\neg(\exists X)\, F &\equiv (\forall X)\, \neg F \\
((\forall X)\, F \wedge (\forall X)\, G) &\equiv (\forall X)\, (F \wedge G) \\
((\exists X)\, F \vee (\exists X)\, G) &\equiv (\exists X)\, (F \vee G) \\
(\forall X)\, (\forall Y)\, F &\equiv (\forall Y)\, (\forall X)\, F \\
(\exists X)\, (\exists Y)\, F &\equiv (\exists Y)\, (\exists X)\, F \\
((\forall X)\, F \wedge G) &\equiv (\forall X)\, (F \wedge G), \quad \text{if } X \text{ does not occur free in } G. \\
((\forall X)\, F \vee G) &\equiv (\forall X)\, (F \vee G), \quad \text{if } X \text{ does not occur free in } G. \\
((\exists X)\, F \wedge G) &\equiv (\exists X)\, (F \wedge G), \quad \text{if } X \text{ does not occur free in } G. \\
((\exists X)\, F \vee G) &\equiv (\exists X)\, (F \vee G), \quad \text{if } X \text{ does not occur free in } G.
\end{aligned}$$

Forma Standard (Standardizing Apart)

registrazione del 13/11/09 alle 09.05

- Si ha la forma standard quando **non ci sono in F due quantificatori che operano sulla stessa variabile, e non ci sono variabili che occorrono sia libere e che limitate**.
- Per ogni formula, esiste una formula semanticamente equivalente dove le variabili sono standardizzate
- Il formato standard serve ad eliminare le ambiguità dalle formule.

Proposizione ausiliaria:

- sia F una formula del tipo F = (QX)G
- sia Y una variabile non presente in F
- E' valido scrivere F ≡ (QY)G{X→Y}

La forma standard si può ottenere per qualsiasi formula

Dimostrazione:

Proviamo una dimostrazione per induzione su quella che può essere la formula più semplice: un atomo.

- Q(x) P[x,y] (dove Q è l'insieme dei quantificatori)
  - Basta cambiare la variabile libera all'interno di P[x,y] (e in tutte le sottoformule) usata dal quantificatore o cambiare lo scope (utilizzando la proposizione ausiliaria).
  - **Le variabili devono essere o libere o quantificate**

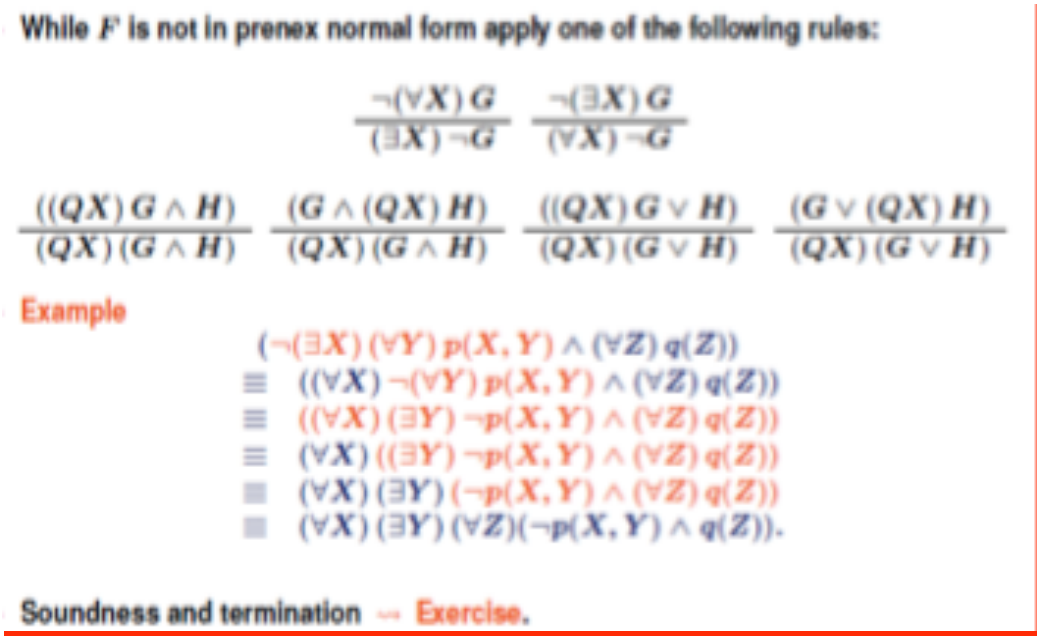
Forme Normali Premesse

Nel calcolo degli enunciati le forme normali sono dei "pacchetti" collegati tutte da ^ .

Esempi di Forma Normale Premessa:

- ∀x ∃y *[P(x,y) ^ Q(z) ^ K(y)]:* questa formula ha tutti i quantificatori posti a sinistra
- ∀x (P(x) ^ ∃y Q(y)): i quantificatori non sono tutti avanti, quindi **NON** è in forma premessa

Algoritmo per la trasformazione in forma premessa



In queste formule G è detta Matrice della formula F

Formula alla lavagna:
- ∀x ∃y [P (x,y)]
- I : {x,y} → $\mathbb{N}$
- x = 5 ∃y P(5,y)
- y : 5 → valore$_5$ ∈ $\mathbb{N}$ I (P(5, valore$_5$)) = ⊤
- y : 6 → valore$_6$ ∈ $\mathbb{N}$ I (P(6, valore$_6$)) = ⊤
- Scrivo in modo analogo: ∀x [P (x,g(x))]

Quest'ultima forma è analoga alla prima. In questo modo elimino gli "esistenziali" sostituendo a ciascuno di essi una funzione (un funtore).

In questo modo sto cercando di eliminare i quantificatori in favore di una matrice. In questo modo, provando la falsificabilità della matrice, potrò dimostrare veridicità della funzione iniziale.

Forma Normale di Skolem

- Il nostro obiettivo fondamentale è quello di eliminare i quantificatori esistenziali.
- Dato che ogni volta che troviamo un quantificatore lo sostituiamo con un funtore, abbiamo bisogno di avere un insieme di funtori diversi tra loro che possano essere usati per sostituire tutti i vari quantificatori. (In pratica ci servono dei nuovi simboli, esterni al linguaggio, che ci servono per eliminare i qualificatori esistenziali)
- Questi nuovi simboli verranno chiamati **Simboli di funzione di Skolem**
- Serviranno anche nuove costanti chiamate **Simboli costanti di Skolem**

Definizione: una formula è in forma normale di skolem se è nella forma (∀X1)…(∀Xn)G dove n≥0 e X1,…,Xn sono variabili e G non contiene alcun ulteriore quantificatore.

Trasformazione nella forma normale di Skolem

Sia F una formula in forma normale premessa (con le variabili tutte standardizzate)

Finché F non è in forma normale di Skolem, applica la seguente regola:

$$\frac{(\forall X_1)\ldots(\forall X_n)\,(\exists Y)\,G}{(\forall X_1)\ldots(\forall X_n)\,(G\{Y\rightarrow g(X_1,\ldots,X_n)\})}$$

Teorema: Se G è la forma normale di skolem della formula F, allora F è soddisfacibile sse G è soddisfacibile.

Forma Clausale

- In tutti i passaggi precedenti ho sempre conservato la validità.
- Ho quindi una formula nella forma $\forall G \equiv (\forall X_1) \ldots (\forall X_n)G$, dove $X_1,\ldots,X_n$ sono variabili presenti in G.
- Nella matrice G non ho quantificatori
- Posso tralasciare tutti i quantificatori e trasformare G in forma clausale effettuando la semplificazione con le regole di Gentzen.

Regole di Gentzen

Caso	Operazioni sul vero	Operazioni sul falso
∧	$\dfrac{\Gamma, a, b \vdash \Delta}{\Gamma, [\, a \wedge b \,] \vdash \Delta}$	$\dfrac{\Gamma \vdash a, \Delta \qquad \Gamma \vdash b, \Delta}{\Gamma \vdash [\, a \wedge b \,], \Delta}$
∨	$\dfrac{\Gamma, a \vdash \Delta \qquad \Gamma, b \vdash \Delta}{\Gamma\, [\, a \vee b \,] \vdash \Delta}$	$\dfrac{\Gamma \vdash a, b, \Delta}{\Gamma \vdash [\, a \vee b \,]\, \Delta}$
→	$\dfrac{\Gamma \vdash a, \Delta \qquad \Gamma, b \vdash \Delta}{\Gamma, [\, a \rightarrow b \,] \vdash \Delta}$	$\dfrac{\Gamma, a \vdash b, \Delta}{\Gamma \vdash [a \rightarrow b], \Delta}$
¬	$\dfrac{\Gamma \vdash a\, \Delta}{\Gamma, \neg a \vdash \Delta}$	$\dfrac{\Gamma\ , a \vdash \Delta}{\Gamma \vdash \neg a, \Delta}$
∀	$\dfrac{\Gamma, \forall x\, A(x), A(a_0) \vdash \Delta}{\Gamma, \forall x\, A(x) \vdash \Delta}$	$\dfrac{\Gamma \vdash A(a), \Delta}{\Gamma \vdash \forall x\, A(x), \Delta}$
∃	$\dfrac{\Gamma\ , A(a) \vdash \Delta}{\Gamma, \exists x\, A(x) \vdash \Delta}$	$\dfrac{\Gamma \vdash \exists x\, A(x), A(a_0), \Delta}{\Gamma \vdash \exists x\, A(x), \Delta}$

Universo di Herbrand

Dato un insieme S di clausole l'universo di Herbrand H(S) per S è definito come:

- H(S) contiene i simboli di costanti che occorrono in S
- se f è un simbolo di funzione ennaria che occorre in S e h1, …, hn sono elementi di H (S), allora anche f(h1,…,hn) sta in H(S)

Esempio:

se S={p(a),p(X),q(Y),q(f(Y))}

allora H(S) = {a, f(a), f(f(a)), ….}

Base di Herbrand

Dati S e H(S) come prima, la base di Herbrand è l'insieme delle istanze ground elle formule atomiche che occorrono in S

Esempio:
B(S) = {p(a), p(f(a), p(f(f(a),…,q(a), q(f(a)),…..}

Interpretazione all'Herbrand

Interpretazione di un insieme S di clausole in cui:

- il cui dominio è l'universo H(S)

- ogni simbolo costante è intepretato sulla corrispondente costante H(S)
- ogni simbolo di funzione è intepretato come funzione che trasforma h1,…,hn in f(h1, …,hn) (tutti presenti in H(S))
- ogni simbolo di predicato è interrpetato su una relazione in B(S)

In pratica definire un'interpretazione di Herbrand corrisponde a dire quale sottoinsieme della base di Herbrand è vera nell'interpretazione.

Interpretazione di Herbrand corrispondente

Un sottoinsieme J dell'universo di Herbrand è chiamata **corrispondente** ad I se e solo se la seguente condizione è valida per tutti i predicati p/n:

$$p(t1,\ldots,tn)^I = T \text{ sse } p(t1,\ldots,tn) \in J$$

Applicazione dell'interpretazione di Herbrand

Un insieme S di clausole è insoddisfacibile se e esolo se non esiste un'interpretazione di Herbrand che lo soddisfa (quindi nel processo di dimostrazione ci si può limitare a considerare tali interpretazioni)

Unificazione

Un problema di unificazione consiste nel cercare, all'interno di un sistema di equazioni, una sostituzione che renda le equazioni al suo interno equivalenti tra loro.

Una funzione σ si dice **unificatore** per due formule o termini E1 e E2, sse σ(E1) = σ(E2).

Due espressioni E1 ed E2 sono unificabili se esiste un unificatore.

Algoritmo di unificazione

Slide 50

Data una formula F formata da n termini E1, E2. In più ho la sostituzione s:

SOST unifica(termine E1,E2; SOST s)

```
if (s==fail) return (fail)
a=E1s            //E1s vuol dire che eseguo la sostituzione s su E1
b=E2s            //idem
if (a e b sono costanti e a==b) return(s)
if (a è una variabile e b un termine e e a non occorre in b) return (s=s+{a→b))
if (b è una variabile e a un termine e b non occorre in a) return (s=s+{b→a))
if (a e b sono termini composti con lo stesso simbolo di funzione e stesso numero di argomenti)
        for (i=1;i≤n; i++)
        {
                s1=unifica(iesimo termine di a, iesimo termine di b, s)
```

```
            s=s+s1
        }
        return (s)
else
        return (fail)
```

Metodi di Dimostrazione

...arrivare alla clausa vuota tramite la trasformazione in clause, sostituzione e unificazione.

The Factoring Rule

registrazione 16/11/09 alle 09.05

Soundness and Completeness Theorem

registrazione 19/11/09 alle 09.05 l Slide pagina 73

Lifting Lemma

Col lifting Lemma cerco una sostituzione che mi permette di costruire una contraddizione (un “controesempio”) nelle formule che voglio verificare.

Questo controesempio sarà dato da delle sostituzioni fatte appositamente per le formule che voglio falsificare. Una volta falsificata una delle formule, ho ottenuto ciò che volevo.

Solitamente si cerca di sostituire in modo che variabili del tipo -p(a) e ¬p(X) si possano annullare a vicenda, per esempio con la sostituzione $\sigma = \{ X \rightarrow a \}$

slide 83: Esempio ricapitolativo l Ascolta alle 9.37

Ricorsività

Registrazione alle 10.19

In questa parte si intende $\top = 0$ e $\bot = 1$ solo per convenzione. Potrei benissimo fare il contrario ma ciò comporterebbe il cambio delle operazioni che effettuo tra i predicati. Se invertissi i valori, per esempio, tutte le operazioni di ^ diverrebbero operazioni di v e viceversa.

Ricorsività ad alti tipi

registrazione del 20/11/09 alle 09.00

E' quel metodo che ci permette di creare una funzione contenente delle variabili. A queste variabili posso sostituire valori oppure altre funzioni.

Quindi ottengo programmi da altri programmi.

Ciò comporta che il mio programma originale (la funzione) non ha un output semplice, basato su dei valori in input, ma può restituirmi un nuovo programma (una nuova funzione) quando alle variabili sostituisco altre funzioni.

Funzioni ricorsive primitie

Da cosa è formata una definizione induttiva?

Base:

$\lambda x_0,\ldots, x_n.Z(x_0,\ldots, x_n) = 0$; <-- Funzione Zero

$\lambda x_0,\ldots, x_n.U^i_n (x_0,\ldots, x_n) = x_i$ <-- Funzione Selettrice

$\lambda x.S(x) = x + 1$ <-- Funzione Successore

Passo:

date:

$\lambda x_0,\ldots, x_n.g(x_0,\ldots, x_n)$ <-- Funzione Ricorsiva

$\lambda x_1,\ldots, x_m.f_0(x_1,\ldots, x_m)$,……, $\lambda x_1,\ldots,x_m.f_n(x_1,\ldots, x_m)$ <-- Composizione

Si ottiene per composizione

$\lambda x_0,\ldots, x_n.g(f_0(x_1,\ldots, x_m),\ldots\ldots,f_n(x_1,\ldots, x_m))$ <-- Ricorsione

Ricorsione

E' formata da una **base** (g)

$$f\ (0,\ x_1,\ \ldots\ x_n)\ =\ g\ (x_0,\ \ldots,\ x_n)$$

dove $x_1,\ldots,x_n$ rappresenta ~~i vincoli~~ gli input al passo corrente

Da una funzione h che mi restituisce il valore della funzione tenendo conto del risultato del passo precedente all'attuale

$$h(y,z,x_1,\ldots,x_n)$$

dove y è il passo in cui mi trovo (n) e z è il risultato del passo precedente.

e da un **passo**

$$f\ (n+1,\ x_1,\ \ldots\ ,\ x_n)\ =\ h\ (\ y,\ f(n,x_1,\ldots,x_n),\ x_1,\ldots,x_n)$$

Esempio:

FIB (0) = 0

FIB (1) = 1

FIB (n+1) = + (FIB(n), FIB(n-1))

Funzioni ricorsive primitive

Errori slide 8:

Nella costruzione della ricorsione della somma , al primo rigo dovrebbe esserci a[=U^1_2(a,0)] non U^0_2

il rigo +(a,3)=S(+(a,2)=...=...=0 è sbagliato. Alla fine c'è =a non =0.

Errori slide 10:

il minimo si calcola come a-(a-b) non come a-(b-a)

Errori slide 11

nonsg(a) = 1 se a diverso 1, non da 0

Vedi gli algoritmi sulle slide

registrazione alle 10.21

Successore

- S(n) = n + 1

Somma

- +(a, 0) = a
- +(a, n+1) = S (+(a,n))

Esempio:
+(a,3) = S(+(a,2)) = SS(+(a,1)) = SSS(+(a,0)) = SSS(a)

Prodotto

- *(a, 0) = 0
- *(a, n+1) = +(a, *(a,n)) = S(a, *(a,n))

Esempio:

*(a,3) = +(a, *(a,2)) = +(a, +(a, *(a,1))) = +(a, +(a, +(a, *(a,0)))) = +(a, +(a, +(a, 0)))

Esponente

- esp(a, 0) = 1
- esp(a, n+1) = * (a, esp(a,n))

Esempio:

esp(a,2) = *(a, esp(a,1)) = *(a, *(a,esp(a,0))) = *(a, *(a,1))

Precedente

- pre(0) = 0
- pre(n+1) = n

Fattoriale

- fatt(0) = 1
- fatt(n+1) = *(n+1, fatt(n))

Sottrazione chiusa

÷(a, b) = a - b se b ≤ a

÷(a, b) = 0 se b > a

- ÷(a, 0) = a
- ÷(a,n+1) = pre (÷(a,n))

Minimo tra a e b

min(a,b) = b ÷ (a ÷ b)

min(a0, a1, … , an) = min(an, (min(…,min(a1,a0))))

Massimo tra a e b

max(a,b) = a + b ÷ (a,b)

max(a0,a1, … ,an) = max(an, max(…,max(a1,a0)))

Non-Segno di a

Semantica:

nonsg(1) = 0 se a = 1
nonsg(0) = 1 se a ≠ 1

Definizione:

nonsg(a) = ÷(1,a)

Segno di a

sg(a) = 1 ÷ nonsg(a)

ossia:

sg(0) = 0

sg(1) = 1, sg(2) = 1, sg(3)=1, ...

Valore assoluto

| a - b | = ÷(a, b) + ÷(b, a)

Resto

- rm(a, 0) = 0
- rm(a, n+1) = (rm(a, n) + 1) * sg(l a÷(rm(a,n) +1) l)

Divisione

Questo metodo calcola quante volte ottengo zero nel procedimento di calcolo del resto:

- div(a,0) = 0
- div(a,n+1)= div(a,n) + nonsg(la ÷ (rm(a,n)+1)l)

Dimostrazione che la Sommatoria è una funzione ricorsiva primitiva

Sia la funzione g

$$g(x,z) = \sum^{z}_{y=0} f(x,y,z)$$

una funzione ricorsiva primitiva

Si definisce la funzione

$$g^*(x,0,z) = f(x,0,z)$$
$$g^*(x,w+1,z) = f(x,w+1,z) + g^*(x,w,z)$$

Nota che:

$$g^*(x,w,z) = \sum^{w}_{y=0} f(x,y,z)$$

quindi

$$g(x,z) = g^*(x,z,z)$$

Il nostro obiettivo è quello di esprimere tramite la ricorsione la logica del prim'ordine

Predicati (ricorsivi primitivi)

registrazione 27/11/09 alle 9.05

Definizione: sono funzioni che danno in output solo vero o falso.

Un predicato è ricorsivo primitivo se esiste una funzione ricorsiva f a cui sono date in input le n variabili del predicato e restituisce:
- 0 se il predicato è vero
- 1 se il predicato è falso.

Questa è detta **funzione caratteristica**.

Vogliamo dimostrare che tutti i connettivi sono ricorsivi primitivi.

Dati P e Q due predicati ricorsivi le cui funzioni caratteristiche sono f_p e f_q.

- **AND: ^**
  P ^ Q = sg(f_P + f_Q)
  in questo modo definiamo l'and come una funzione ricorsiva primitiva attraverso la composizione delle due funzioni ricorsive primitive *sg* e *+*.

- **OR: v**
  P v Q = sg(f_P * f_Q)
- **NOT: ¬**
  ¬P = nonsg(f_P)
  basta sostituire nelle precedenti *nonsg* a *sg*.
- **IMPLICAZIONE: →**
  Basta ricordare che → = ¬ A v B

Grande E e grande O

Errore slide 17: sommatoria e produttoria sono invertite

- Grande O: $\vee^{n}_{i=0}$ P(i) = P(0) v P(1) v … v P(n)
- Grande E: $\wedge^{n}_{i=0}$ P(i) = P(0) ^ P(1) ^ … ^ P(n)

Se P è un predicato ricorsivo primitivo allora:

- $V^{n}_{i=0}$ P(i) è ricorsivo primitivo sg($\prod^{n}_{i=0} f_P(i)$)
- $\wedge^{n}_{i=0}$ P(i) è ricorsivo primitivo sg($\sum^{n}_{i=0} f_P(i)$)

Quantificatori limitati

Si introducono i simboli ∀ e ∃ che possono essere definiti come:

$$\forall(0 <= i <= n)\ P(i) = \wedge^{n}_{i=0}\ P(i)$$
$$\exists(0 <= i <= n)\ P(i) = \vee^{n}_{i=0}\ P(i)$$

si dicono limitati perché sono racchiusi tra 0 e n

Essere numero primo

Un numero primo si definisce tale se è divisibile per 1 e per se stesso.

Escludendo 0 e 1, definiamo x primo, se per ogni numero compreso da 1 (escluso) a x-1, non trovo un numero per cui è divisibile.

In pratica un numero è primo se:

$$(x\neq 0)\ \wedge\ (x\neq 1)\ \wedge\ \wedge^{x}_{i=2}\ nonsg(rm(x,i))$$

Ricorda anche che intendiamo 0 = ⊤ e 1 = ⊥, quindi dire "0 ^ 0 ^ 0 …" ≡ "⊤ ^ ⊤ ^ ⊤ …"

Altre funzioni (≠,≤,divisibile,pari,dispari)

- $x\neq y = nonsg(|x\div y|)$
- $x\leq y = \vee^{y}_{i=0}(x+i=y)\ [\exists 0\leq i\leq y(x+i=y)]$
- $divisibile(x,y) = \vee^{y}_{i=0}(x*i=y)$
- $pari(y) = divisibile(2,y)$
- $dispari(y) = nonsg(pari(y))$

Minimalizzazione limitata

La minimalizzazione si esprime con μ e si dice μz, per esempio, per esprimere "il minimo z"

Data una funzione $g(x_0,\ldots,x_n) = y$ si dice che g è minimalizzazione di P se $P(x_0,\ldots,x_n,y) = 0$

$$g(x_0,\ldots,x_n) = \mu y(0 \leq y \leq n) : P(x_0,\ldots,x_n,y) = 0 \quad \text{(dove } 0 \equiv \top\text{)}$$

Se tale y non esiste, allora y = 0

Per individuare la y minima si utilizzano le funzioni *sg*, *+*, ***, $\sum$, $\prod$.

Ricordando che: sg(a) = 0 se a=0, individuiamo y come segue:

$$sg(P(x_0,\ldots,x_n,0)) +$$
$$sg(P(x_0,\ldots,x_n,0)) * sg(P(x_0,\ldots,x_n,1)) +$$
$$sg(P(x_0,\ldots,x_n,0)) * sg(P(x_0,\ldots,x_n,1)) * sg(P(x_0,\ldots,x_n,2)) +$$
$$\ldots * \ldots +$$

Ovvero il minimo y che azzera f è:

$$\sum^{n}_{y=0} \prod^{y}_{j=0} sg(P(x_0,\ldots,x_n,j))$$

Se $P(x_0,\ldots,x_n,y)$ è è un predicato ricorsivo primitivo allora la funzione ottenuta dall'applicazione dell'operatore μ è ricorsiva primitiva.

Come funziona?
Nota come l'algoritmo è disposto in righe: ad ogni riga c'è una serie di prodotti; ogni riga viene sommata con la successiva. Il risultato di ogni riga è 1 o 0 perché vengono moltiplicati tra loro i risultati di ogni operazione sg(x) (che può restituire solo 0 o 1).
In ogni riga, all'interno del predicato viene cambiata una variabile in input che scorre da 0 a y. Ad ogni riga successiva, c'è un'ulteriore moltiplicazione di un predicato con la variabile che scorre, in questa riga, da 0 a y+1 rispetto alla precedente.
Così facendo si arriverà ad un punto in cui l'inserimento di un particolare valore in quella variabile, azzererà il predicato, portando ad una moltiplicazione per zero in tutta la riga (ricorda che sg(0)=0). Quindi anche sommando altre righe successive, questo predicato sarà sempre presente e le azzererà tutte, lasciando che la somma sia sempre pari al valore della y minima che stiamo cercando.

Applicazione della minimalizzazione: Enumerazione dei numeri primi

Definiamo p la proprietà che esprime l'essere numero primo.

$$p(1) = 2$$
$$p(n+1) = \mu z(\, p(n)+1 \leq z \leq p(n)! + 1) : z \text{ è primo}$$

per verificare che z sia primo si utilizza quanto detto al paragrafo "Essere numero primo"

Applicazione della minimalizzazione: Fattore con massimo esponente

Dato un numero *x* e la sua scomposizione in fattori primi, vogliamo sapere qual'è il massimo esponente di uno dei fattori scelto. Di questo fattore dobbiamo indicare la posizione in ordine e la chiamiamo *n* (Es: scomp(20) = $2^2 * 5$, quindi n=1 indica 2, invece n=2 indica 5).

$$(x)_n = \mu z(1 \leq z \leq n) : \neg div(p(n)^{z+1}, x)$$

Ricorda che p(n) è la funzione che restituisce l'n-esimo numero primo.

Esempio:

Dato x = 20, la sua scomposizione è pari a $2^2 * 5$.

Scegliamo n = 1 (la posizione del fattore)
Il nostro $(x)_n$ sarà 2, ossia l'esponente del fattore 2 nella fattorizzazione.

Esecuzione:

$$\neg div(p(1)^{0+1},20) \equiv \neg div(2^1,20) \equiv \neg 0 = 1 +$$
$$\neg div(p(1)^{1+1},20) \equiv \neg div(2^2,20) \equiv \neg 0 = 1 +$$
$$\neg div(p(1)^{2+1},20) \equiv \neg div(2^3,20) \equiv \neg 1 = 0 =$$
$$= 2$$

Quindi il massimo esponente del fattore numero n=1 della scomposizione di x=20 è $(20)_1 = 2$.

Calcolo della radice

√x = y significa trovare il minimo y tale che elevato al quadrato mi dia x. In questo caso noi troveremo

$$\mu z \ (0 \leq z \leq x) : (z+1)^2 > x$$

Calcolo del logaritmo

$\log_2 a = b$ significa trovare il minimo b tale che 2 elevato a questo minimo sia maggiore di a:

$$\mu z \ (0 \leq z \leq a) : 2^{z+1} > b$$

Il problema dell'esiste ∃

Non sono in grado di lavorare con ∃ e ∀, quindi devo reinterpretarli.

∃ è come dire: 0 <= z <= y * x + z = y

per dubbi senti la registrazione alle 9.25

Compattare serie di numeri

Se ho una serie di numeri <a,b,c> posso compattarli nella forma $2^a * 3^b * 5^c = d$

Se voglio tornare da *d* alla serie di <a,b,c> mi basta scomporre d in numeri primi d e otterrò 2 ripetuto a volte, 3 ripetuto b volte, 5 ripetuto c volte e così via. Ovviamente 2,3,5 e successivi sono tutti numeri primi.

Eulero: p(n) < p(n+1) < p(n)! + 1 dove p(n) è un numero primo

Ricorsione sul decorso dei valori

registrazione del 30/11/09 alle 09.10 I Slide 22

Definiamo due funzioni: g(x0,…,xn) e h(x0,…,xn,y,z0,…,zk) ricorsive primitive. Quindi per calcolare h abbiamo bisogno del passo e dei risultati precedenti (i parametri z0,…,zk). Per poter calcolare i risultati precedenti, ho necessità di avere $n_i(x_0,\ldots,x_k)$ con $0 \leq i \leq k$ funzioni. La particolarità di queste funzioni è che il loro output è sempre inferiore al loro input: ∀i ∀m $n_i(m) \leq m$.

Possiamo quindi definire la funzione (ricorsiva primitiva) f^0 **tramite decorso dei valori**

$f^0(x_0,\dots,x_n,0) = g(x_0,\dots,x_n)$

$f^0(x_0,\dots,x_n,n+1) = h(x_0,\dots,x_n,n,f^0(x_0,\dots,x_n,n_0(n)),\dots,f^0(x_0,\dots,x_n,n_k(n)))$

Definiamo la funzione:

$f(x_0,\dots,x_n,0) = 2^{g(x0,\dots,xn)}$; $[= p(1)^{g(x0,\dots,xn)}]$

$f(x_0,\dots,x_n,n+1) = f(x_0,\dots,x_n,n) * p(n+2)^a$ dove:
$a = h(x_0,\dots,x_n,n,(f(x_0,\dots,x_n,n)_{n0(n)}),\dots,(f(x_0,\dots,x_n,n)_{nk(n)}))$

Si prova che:

$\forall(x_0,\dots,x_n)\forall n : f^0(x_0,\dots,x_n,n) = f(x_0,\dots,x_n,n)_{p(y)}$

$n=0 \quad f(x_0,\dots,x_n,0) = 2^{g(x0,\dots,xn)}$

$(f(x_0,\dots,x_n,0))_1 = (2^{g(x0,\dots,xn)})_1 = g(x_0,\dots,x_n) = f^0(x_0,\dots,x_n,0)$

Passo:

$f(x_0,\dots,x_n,n+1) = f(x_0,\dots,x_n,n) * p(n+2)^l$ dove
$l = (x_0,\dots,x_n,n,(f(x_0,\dots,x_n,n)_{n0(n)}),\dots,(f(x_0,\dots,x_n,n)_{nk(n)}))$

Quindi:

$f^0(x_0,\dots,x_n,n+1) = (f(x_0,\dots,x_n,n+1))_{p(y+2)} = h(x_0,\dots,x_n,n(f(x_0,\dots,x_n,n)_{n0(y)}),\dots,(f(x_0,\dots,x_n,n)_{nk(n)}))$

Per ipotesi induttiva si ha:

$h(x_0,\dots,x_n,n,(f^0(x_0,\dots,x_n,n0(n)),\dots,(f^0(x_0,\dots,x_n,nk(n))) = f^0(x_0,\dots,x_n,n+1)$

Funzioni Totali

Sono le funzioni che, dato un elemento del dominio, mi danno un elemento del codominio. Per ogni elemento in input, restituiscono un elemento in output.

Una funzione totale converge sempre (ha sempre un risultato).

Metodo Diagonale

Slide 25

Data una qualsiasi successione di funzioni totali, esiste una funzione totale che non appartiene alla successione.

In pratica se prendo un'insieme di funzioni totali, comunque lo prendo, ce ne sarà sempre almeno una che è una funzione totale ma che manca nell'insieme.

Prendo le funzioni

$$\lambda x.P_0,\ \lambda x.P_1,\ \dots,\ \lambda x.P_n,\dots$$

Considero la funzione:

$$k(x) = P_x(x) + 1$$

Considero λy.k è una generica funzione totale della successione tale che:

$$\forall x\ \lambda y.k \neq \lambda x.P_x$$

Infatti

$$\forall x\ k(x) \neq P_x(x)$$

essendo

$$k(x) = P_x(x)+1$$

Quindi k non è un elemento della successione

Questo avviene in quanto la successione è formata da funzioni $P_x(x)$ che calcolano un risultato in uno specifico codominio. k(x), ha bisogno di una funzione che calcoli $P_x(x)$ e che poi sommi il risultato a 1! La differenza è sottile, ma fondamentale. k(x) ha bisogno di una funzione universale che calcoli P_x e questa funzione non fa parte delle funzioni totali.

Applicazioni del metodo diagonale

Le funzioni ricorsive primitive sono totali (ossia sono definite su ogni elemento del dominio).

Esiste una funzione calcolabile ma non ricorsiva primitiva?

Dimostrazione per assurdo:

- Si considerano tutte le funzioni ricorsive primitive unarie [$\lambda x.P_x$]
- Si considera la funzione $k(x) = P_x(x) + 1$ che non fa parte della successione delle ricorsive primitive.
- $\lambda x.k$ è calcolabile e unaria
- Ma $\lambda x.k$ è differente da tutte le altre funzioni ricorsive primitive
- Per calcolare k(x) trovo il programma $\lambda x.Px$
- Calcola $P_x(x)$ e sommo 1 al risultato
- Quindi ho trovato una funzione calcolabile, che però non fa parte dell'insieme delle ricorsive primitive *(**assurdo**)*

Funzioni ricorsive parziali

E' un linguaggio per simulare varie funzioni con le funzioni di base (zero, selettore, successore) effettuando operazioni di composizione, ricorsione e minimalizzazione illimitata.

A causa della minimalizzazione illimitata queste funzioni posso non convergere e quindi vengono definite parziali in quando non appartenenti alle totali (non sono definite su ogni elemento del dominio).

Funzione Universale (da controllare)

Capitolo del Catland "Secondo teorema della ricorsione - Kleene"

Registrazione del 3/12/2009 alle 09.05

- Con Y_u (x,y) indichiamo una funzione che è equivalente ad un calcolatore: gli si da in pasto un programma ne restituisce un risultato (forse)
  - con x intendo il programma
  - con y intendo il dato su cui il programma lavorerà
- Occhio che per y intendo …

- Ho una funzione $FI_{f(x)}$ che mi crea una permutazione di ….

- So che per $FI_{f(n)}$ avrò un n per cui $FI_{f(n)} = f(n)$
- Considero la funzione FI *f($FI_x(x)$)* dove tutto *f($FI_x(x)$)* è l'indice di FI
- Calcolo l'indice --> f ($FI_x(x)$) = f ($FI_{10}(10)$) = f (100) = 1000 => e quindi avrò: FI_{1000}
- FI è una funzione ricorsiva quindi ha un indice: m
- Posso scrivere $FI_m(x) = f (FI_x(x))$

Funzioni Universali

slide 31

Una funzione universale ϕU prende in input l'indice y di una funzione ricorsiva parziale più i dati della funzione e mi restituisce in output il risultato della funzione di indice y

$$\varphi U(y, x_0, \ldots, x_1) \approx \varphi_y(x_0, \ldots, x_n)$$

Un compilatore è una funzione universale!

Teorema s-m-n

Data λy1,…,yn,z1,…,zm.ϕx che ha in input *n+m* variabili, esiste un algoritmo ricorsivo ϕ_s che fornisce come output un programma di *m* variabili, il quale fornisce gli stessi risultati di ϕ_x e che codifica le rimanenti *n* variabili:

$$\lambda y1, \ldots, yn, z1, \ldots, zm.\varphi_x \approx \lambda z1, \ldots, zm.\varphi_{s(x,y1,\ldots,yn)}$$

Quindi ho la funzione ϕ_s che lavora su *m* variabili, mentre ϕ_x lavora su n + m variabili. Bloccando alcuni parametri di ϕ_x (gli y) posso ottenere che la funzione ϕ_s si comporta esattamente come ϕ_x quando il parametro bloccato rimane sempre lo stesso.

Esempio:

$$\lambda z, y.\varphi_x \text{ es } +(z, y)$$
$$\lambda z.\varphi_s \text{ es } \varphi_s(z) = \varphi_{(s(x,y))}(z)$$

blocco y=3 nella funzione di indice x=1 (quindi, per esempio, la somma) e faccio variare z:

$$\varphi_{s(1,3)}(z) = \varphi_{(x=1)}((y=3), z)$$

In pratica l'operazione che farò sarà la somma (perché per ipotesi ha indice 1) e la eseguirò nella forma +(3,z).

+ (funzione di indice 1)	y=1	y=2	y=3
z=1	2	3	4
z=2	3	4	5
z=3	4	5	6
z=4	5	6	7

Primo problema dell'Alt (*)

Data una **successione delle funzioni ricorsive** ϕ_0, ϕ_1, ϕ_2 **non esiste la funzione ricorsiva che dia un risultato nel caso in cui una delle funzioni della successione diverge.**

Dimostrazione per assurdo:

Sia $\varphi(x,y)$ la funzione in grado di calcolare se la funzione di indice x diverge. φè così definita

$$\varphi(x,y) = \begin{cases} -\ 1 \text{ se } \varphi_x(y) \text{ converge} \\ -\ 0 \text{ se } \varphi_x(y) \text{ diverge} \end{cases}$$

Ora sia f(x,y) una delle funzioni della mia successione che sfrutta $\varphi(x,y)$ per ottenere un risultato. f è così definita:

$$f(x,y) = \begin{cases} -\ 1 \text{ se } \varphi(x,y)=0 \\ -\ \text{oppure f diverge} \end{cases}$$

Ora si calcola un caso particolare di f in cui x=e, dove e è l'indice della funzione f stessa.

Si supponga che il risultato di $\varphi(e,y)$ sia 0, il che significa che la funzione di indice e (ossia f) **diverge**; accadrà, però che se $\varphi(x,y) = 0$, allora f(e,y) = 1, quindi **converge**!! **Assurdo!**

$$f(e,y) = \begin{cases} -\ 1 \text{ se } \varphi_e(y) \text{ diverge} \\ -\ \text{oppure f diverge} \end{cases}$$

Secondo teorema della ricorsione (Kleene): punto fisso

Per ogni funzione f totale unaria e ricorsiva, esiste un n tale che

$$\lambda y_0,\dots,y_k.\Phi_{f(n)} \approx \lambda y_0,\dots,y_k.\Phi_n$$

Dimostrazione:

Si considerino le due funzioni:

$$\Phi_{f(\Phi x(x))}(y_0,\dots,y_k) \approx \Phi_{\Phi m(x)}(y_0,\dots,y_k)$$

$\Phi_m(x)$ è la funzione $f(\Phi_x(x))$,quindi m è l'indice della funzione f e quindi

$$f(\Phi x(x)) = \Phi(m(x))$$

Quindi ponendo x=m otterrò:

$$f(\Phi m(m)) = \Phi(m(m))$$

Ora si consideri $n = \varphi_m(m)$ otterrò:

$$f_{f(n)}(y_0,\dots y_k) = \Phi_n(y_0,\dots y_k)$$

Ecco cosa accadrà:

$\Phi_{f(n)}(y_0,\dots y_k) \approx$
$\varphi_U(f(n),y_0,\dots y_k) \approx$
$\varphi_U(f(\Phi m(m)),y_0,\dots y_k) \approx$ //Ricorda che n = ϕ_m(m)
$\varphi_U(f(f(\Phi m(m))),y_0,\dots y_k) \approx$
$\varphi_U(f(f(f(f(\dots f(\Phi m(m)))))),y_0,\dots y_k)$

Punto fisso: dimostrazione grafica (*)

Per ogni k chiamiamo E_k la sequenza di funzioni ricorsive enumerate dalla funzione ϕ_k.

Costruiamo la matrice:

E_0	$\phi_{\phi_0(0)}$	$\phi_{\phi_0(1)}$	$\phi_{\phi_0(2)}$	....	$\phi_{\phi_0(k)}$
E_1	$\phi_{\phi_1(0)}$	$\phi_{\phi_1(1)}$	$\phi_{\phi_1(2)}$	....	$\phi_{\phi_1(k)}$
E_2	$\phi_{\phi_2(0)}$	$\phi_{\phi_2(1)}$	$\phi_{\phi_2(2)}$	....	$\phi_{\phi_2(k)}$
...	...	...	...	...	...
E_k	$\phi_{\phi_k(0)}$	$\phi_{\phi_k(1)}$	$\phi_{\phi_k(2)}$	...	$\phi_{\phi_k(k)}$
...	...	....	....	....	....

La lista $E_0,E_1,\ldots$ include tutte le possibili enumerazioni delle funzioni calcolabili (ossia include tutti i possibili argomenti che è possibile dare ad ogni funzione)

Ora si considera l'enumerazione ottenuta dalla diagonale.

$$D: \Phi_{\Phi 0(0)}, \Phi_{\Phi 1(1)}, \Phi_{\Phi 2(2)}, \ldots, \Phi_{\Phi k(k)}, \ldots$$

ora se abbiamo una funzione f totale e ricorsiva possiamo definire:

$$D^*: \Phi_{f(\Phi 0(0))}, \Phi_{f(\Phi 1(1))}, \Phi_{f(\Phi 2(2))}, \ldots, \Phi_{f(\Phi k(k))}, \ldots$$

anche D* è un'enumerazione e quindi esisterà un $E_m = D^*$.

Φ_m è totale, quindi:

E_0	$\phi_{\phi_0(0)}$	...	....	....	....
E_1	...	$\phi_{\phi_1(1)}$	....	....	....
E_2	....	....	$\phi_{\phi_2(2)}$	....	....
...	...	...	...	...	...
$D^* = E_m$	$\phi_{f(\phi_0(0))}$	$\phi_{f(\phi_1(1))}$	$\phi_{f(\phi_2(2))}$	...	$\phi_{f(\phi_m(m))} = \phi_{\phi_m(m)}$
...	...	....	....	....	....

cosi $n = \phi_m(m)$ è il punto fisso.

Così presa la colonna con indice m, avremo che

$$\Phi_{f(\Phi m(m))} = \Phi_{\Phi m(m)}$$

Quindi $\Phi_{\Phi m(m)}$ è uguale ad *n* che è il nostro punto fisso.

Decidibilità

Diciamo che un predicato P è decidibile

- Sse per ogni x del dominio esiste un algoritmo per decidere se $x \in P$ o $x \notin P$ [se cioè ho una procedura che in un numero finito di passi termina dicendomi se x appartiene a P, o in caso contrario, che x non appartiene a P].
- Sse esiste una funzione ricorsiva tale che P(x)=1 se $x \in P$ e P(x)=0 se $x \notin P$. (Ma questa funzione è esattamente la funzione caratteristica di P, perciò:)
- Sse la sua funzione caratteristica è una funzione ricorsiva[2].

Teorema di Rice

1. Consideriamo una qualunque enumerazione delle funzioni ricorsive (ricordiamo che funzioni ricorsive e calcolabili sono la stessa cosa), in cui la funzione φ_i corrisponde alla i-esima funzione ricorsiva. L'insieme di tutte le funzioni ricorsive lo indichiamo con $R = \{\varphi_x | x \in \mathbb{N}\}$.
2. Consideriamo inoltre **l'insieme P** di **funzioni ricorsive** (formalmente $P \subseteq R$), che esprime una certa proprietà di queste funzioni, nel senso che esso contiene solo quelle funzioni ricorsive che soddisfano la proprietà.
3. Consideriamo infine **l'insieme S**, che contiene gli **indici** (secondo l'enumerazione del punto 1) delle funzioni contenute in P, cioè più formalmente $S = \{x | \varphi_x \in P\}$.

L'insieme S è decidibile se e solo se P è l'insieme vuoto, formalmente $P = \emptyset$, o coincide con l'intera classe delle funzioni ricorsive, formalmente $P = R = \{\varphi_x | x \in \mathbb{N}\}$.
Altrimenti, se P è "non banale", **S è indecidibile**.

Dimostrazione per assurdo:

- Sia P ≠ Ø e P ≠ R
- Sia S decidibile
- Siano i ∈ S e j ∉ S e si consideri la funzione:

$$C(x) = \begin{pmatrix} i & se \quad x \notin S \\ j & se \quad x \in S \end{pmatrix}$$

- la funzione C è totale, quindi per il teorema del punto fisso $\exists e \in \mathbb{N}$ tc. $\varphi_{C(e)} = \Phi_e$
- Per definizione C(e) = i se implica che e ∉ S
- Da ciò segue che la funzione ricorsiva di indice *e* non appartiene a P, ossia: $\varphi_e \notin P$
- Ma per ipotesi noi supponiamo che $i \in S$ da cui segue che $\varphi_i \in P$; considerando che per il teorema del punto fisso $C(e) = i = e$ e che quindi $\varphi_{C(e)} = \varphi_i = \varphi_e$ e avendo detto che $\varphi_i \in P$ e $\varphi_e \notin P$ troviamo l'**assurdo**!

La macchina di Turing

Continua registrazione precedente

Un dispositivo che accede a un nastro (potenzialmente) illimitato diviso in celle contenenti ciascuna un simbolo di un alfabeto fissato, più il carattere ø (cella vuota).

La MDT opera sul nastro con una testina: può leggere o scrivere un carattere in una cella, spostarsi a destra o sinistra.

Registrazione del 04/12/2009 alle 09.05 | Slide 9

Slide 71 -> registrazione al min. 27

In ogni istante la macchina si trova in uno stato, e la computazione evolve attraverso la funzione di transizione:

stato corrente + contenuto della cella su cui ´e la testina =)

nuovo stato + carattere da scrivere + spostamento.

Definizione: Una macchina di Turing deterministica (MDT) è una sestupla M= <, ø, Q, q_0, F, σ> , dove:

Γ è l'alfabeto dei simboli di nastro

ø ∈ Γ è il carattere speciale di cella vuota

Q è un insieme non vuoto e finito di stati

q_0 ∈ Q è lo stato iniziale

F ^incluso Q è l'insieme degli stati finali
σ è la funzione di transizione, definita come:

$$\sigma: (Q-F) \times (\Gamma \cup \{\varnothing\}) \rightarrow Q \times (\Gamma \cup \{\varnothing\}) \times \{d, s, i\}$$

Configurazioni e transizioni di una MDT

Consiste nella configurazione istantanea del contenuto del nastro, della posizione della testina e dello stato corrente.

Una configurazione istantanea si rappresenta con una stringa c = xqy, dove:

1. x ∈ ΓΓ* ∪ {ε} (da ora in poi, L_Γ)
2. q ∈ Q
3. y ∈ Γ*Γ∪ {ø} (da ora in poi R_Γ)

In xqy, xy rappresenta il contenuto della sezione non vuota del nastro, q è lo stato attuale e la testina è su primo carattere di y.

Configurazione iniziale e finale

Una configurazione si dice iniziale se:

- lo stato iniziale è q_0
- il nastro contiene l'input x su lxl celle contigue (e le altre vuote)
- la testina si posiziona sul primo carattere di x

Una configurazione si dice finale se:

- lo stato della macchina è uno stato finale

Funzione di Transizione

Data una configurazione, l'applicazione di funzione di transizione su c, genera una nuova configurazione $\sigma(q,a) = c'$.

Computazione di una macchina di Turing

Definizione: Data una macchina di Turing $M=<\Gamma,\o,Q,q_0,F,\sigma>$ e dato un alfabeto in input $\Gamma \subseteq \sum$, una stringa $x \in \sum^*$è accettata (rifiutata) da M se esiste una computazione di accettazione (rifiuto) con $c_0 = q_0x$.

Problema della Terminazione (problema dell'alt)

Questa definizione implica che la macchina potrebbe anche **non terminare**, questo perché la macchina di Turing può effettuare modifiche sulla stringa che può analizzare.

Il problema della terminazione per le macchine di Turing consiste nella non esistenza di una macchina di Turing deterministica che, data in input una MDT e il suo relativo nastro, restituisca 0 se la macchina termina, 1 in caso contrario.

$$\nexists\ HALT \in MDT\ : \quad \forall \mu \in MDT \quad \forall n \in NASTRO$$

risulta che:

$$HALT(\mu,n) = \begin{cases} -\ 0 \text{ se } \mu(n) \text{ DIVERGE} \\ -\ 1 \text{ se } \mu(n) \text{ CONVERGE} \end{cases}$$

Calcolo di funzioni parziali

Data una funzione $f: \sum^* \rightarrow \sum$ ($\sum \in \Gamma$), si dice che μ calcola f se e solo se $\forall x \in \sum^*$ si ha che:
1. f(x)=y allora esiste una computazione che trasforma la configurazione iniziale q_0x in una nuova con stato finale $c = x\o q_f y$, dove $q_f \in F$.
2. Se la funzione f(x) non è definita, allora non esiste una computazione che termina in uno stato finale per la macchina di Turing

Problema della codifica

Un problema del calcolo delle funzioni con le MDT è che lavorano su un dominio diverso da quello delle funzioni. Ciò porta alla necessità di definire funzioni per la codifica di tali dati.

Calcolabilità secondo turing

Formalizziamo il concetto di calcolo secondo Turing (avendo a disposizione una definizione formale di algoritmo).

Definizione: Un linguaggio è decidibile secondo Turing (Tdecidibile) se esiste una macchina di Turing che lo riconosce.

Definizione: Un linguaggio è semidecidibile secondo Turing (T-semidecidibile) se esiste una macchina di Turing che lo accetta.

Definizione: Una funzione è detta calcolabile secondo Turing (T-calcolabile) se esiste una macchina di Turing che la calcola.

Macchine di Turing multinastro

Definizione: Una macchina di Turing a k≥2 nastri (MTM) è una sestupla $M^{(k)} = <\Gamma, ø, Q, q_0, F, \sigma^k>$, con $\Gamma = U^k_{i=1} \Gamma_i$ è l'unione dei k alfabeti di nastro $\Gamma_1,\ldots, \Gamma_2$.

La funzione di transizione è definita come:

$$\sigma^{(k)} : (Q - F) \times \Gamma^*_1 \times \ldots \times \Gamma^*_k \rightarrow Q \times \Gamma^*_1 \times \ldots \times \Gamma^*_k \times \{d, s, i\}^{(k)}$$

Quindi questa macchina prende in input uno stato con k caratteri e k funzioni di transizione sempre sui k nastri. Restituisce uno stato equivalente.

Il potere computazionale di una MTM è pari al potere computazionale di una MDT a singolo nastro. Ciò significa che possono eseguire gli stessi algoritmi.

Configurazione istantanea

Analogalmente alle MDT, anche per le MDM possiamo avere la configurazione istantanea che è una stringa del tipo:

$$q\#a_1 \uparrow b_1\#a_2 \uparrow b_2\# \ldots \#a_k \uparrow b_k$$

dove ai ∈ ΓiΓ*i U {ε} e bi ∈ ΓiΓ*i U {ø}, con ↑ il simbolo che indica la posizione di ogni testina e # un separatore.

NB: ε = parola vuota

Particolarità:

- Il primo nastro viene di solito definito **nastro di lettura.**
- Gli altri nastri sono usati per le varie operazioni da svolgere sulla stringa.

Configurazione iniziale

Per definire la configurazione iniziale si ha la ncessità di inserire un ulteriore carattere speciale Z_0. Quindi possiamo definire una generica stringa rappresentante una configurazione iniziale come:

$$q_0 \# a_1 \uparrow b_1 \# \ldots \# a_k \uparrow b_k$$

Dove $a_i = \varepsilon$, $b_1 \in \Gamma^*_1$, $b_i = Z_0$ con (i=2, …, k), e $q=q_0$.

Macchina di Turing Multitraccia

Questo tipo di MDT ha un nastro di cui ogni cella è composto da una traccia (ossia un vettore di celle). In questo modo, quando la testina effettua la lettura, legge k simboli.

Questi k simboli sono la **cardinalità** della multitraccia.

Simulazione di una MDT Multitraccia con una MDT a singolo nastro

E' possibile, tramite una funzione ingettiva, far corrispondere l'alfabeto di una MDT multitraccia a quello di una semplice MDT. **In questo modo è possibile simulare una MDT multitraccia attraverso una MDT a singolo nastro,** pur comportando una maggior complessità dell'algoritmo.

Simulazione di una MDM tramite una MDT Multitraccia

Avendo una MDM ad n nastri, è possibile simularla attraverso una MDT Multitraccia a 2 * n tracce.

- Nelle tracce dispari della multitraccia sarà presente il carattere ø e il carattere ↓ che rappresenta la posizione della testina del nastro che è simulato dalla traccia successiva.
- Nelle tracce pari della multitraccia ci sarà il contenuto dei nastri della multinastro.

Abbiamo così simulato il comprtamento di una MDM attraverso una Multitraccia. Da qui, **possiamo simulare il comportamento della Multitraccia con una MDT a singolo nastro.**

Costi

Il costo computazionale dell'algoritmo che simula una MDM con una MDT a singolo nastro grava sia sull'alfabeto che sulle regole di transizione.

Costo sulle regole di transizione: Una MDM con t regole di transizione è equivalente ad una MDT a singolo nastro con t^2 regole di transizione.

Costo sull'alfabeto: Una MDM con un alfabeto Γ_i con $(1 \leq i \leq n)$, dove n è il numero dei nastri, è simulabile attraverso una MDT a singolo nastro dove l'alfabeto è pari a $[\ 2 * \max(\ \Gamma_i\ )\ ]^n$.

Macchine di Turing non deterministiche (MTND)

Una MDND è definita così come una MDT. Cambiano però le regole di transizione, che sono così definite:

$$\sigma : Q \times \Gamma^* \rightarrow P(Q \times \Gamma^* \times \{d,s,i\})$$

dove P è l'insieme delle parti.

In questo caso σ è chiamata **funzione di transizione parziale**.

Una MTND ha lo stesso potere computazionale di quella deterministica, ma solo a condizione che la MDT sia vista con un nastro infinito e un tempo infinito di elaborazione.

E' possibile rappresentare una MTND sia tramite matrice che tramite albero delle transizioni. Questo perché dato uno stato q_0 e lo stesso input, lo stato successivo potremmo spostarci nello stato q_1 o q_2.

Grado di non determinismo

Il grado di non determinismo di una MTND è la massima cardinalità tra le regole di transizione.

Es: Se in un grafo ad albero di una MTND il numero massimo di figli tra ogni elemento che rappresenta uno stato è 2, allora il grado di non determinismo è 2.

Equivalenza tra MDT e MTND

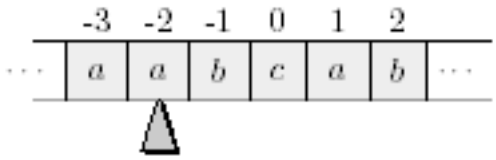
Per ogni MTND esiste una MDT deterministica a 3 nastri equivalente.

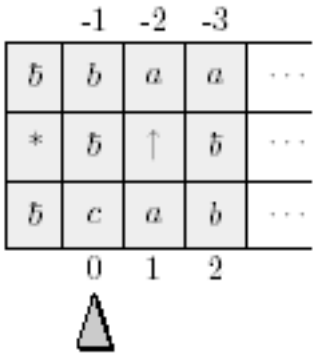
- Nel primo nastro è presente lo stato iniziale della macchina
- Nel secondo ci sono tutte le possibili combinazioni dei cammini dato dal grado di non determinismo della MTND
- Nel terzo nastro avviene l'effettiva computazione delle varie configurazioni

Grazie alla configurazione del secondo nastro, l'albero che definisce la MTND viene analizzata con un'analisi dell'albero in ampiezza (invece che in profondità), così che si sia certi di trovare almeno uno degli stati finali. Tale algoritmo viene applicato così che la macchina deterministica, una volta raggiunto uno stato finale, termini la computazione.

Riduzione delle macchine di turing

Definizione: Per ogni MTD deterministica M (e quindi con nastro infinito), esiste una MDT M' equivalente con nastro semi-infinito a 3 tracce.





vedi slide 56-57-58 slide "capitolo 5"

Per semi infinito si intende che il nastro ha un inizio, ma non ha fine. Un nastro infinito invece non ha né inizio né fine.

- Nella prima traccia della multitraccia è presente la sottostringa a sinistra di una determinata cella di origine della macchina di turing a nastro infinito.
- Nella terza traccia vi è la sottostringa a destra della cella di origine della stessa cella di origine.
- Nella seconda traccia vi è un'isieme di simboli che descrivono lo spostamento della testina
- Alla prima cella della multitraccia è presente una particolare sequenza di simboli che consente di rilevare il passaggio della testina dalla parte sinistra della cella di origine alla relativa parte destra e viceversa. (In pratica definisce se si sta leggendo sulla prima o sulla terza traccia)

Per ogni transizione della MDT a nastro infinito del tipo $\sigma(q_h,a_i) = (q_j,a_k,d)$, ci sono 3 transizioni per la MDT equivalente a nastro semi-infinito. Ogni transizione della macchina a nastro infinito, deve essere emulata con:

- l'operazione di scrittura dei valori
- l'operazione di scrittura del simbolo ↑
- l'operazione di scrittura del simbolo ↓

(Sta cosa è da controllare)

Linearizzazione

Definizione: Per ogni MDT M=<Γ,ø,Q,q_0,F,σ> esiste una MDT M'=<Γ',ø,Q',q'$_0$,F',σ'>, con l Γ'l = 1, equivalente a M.

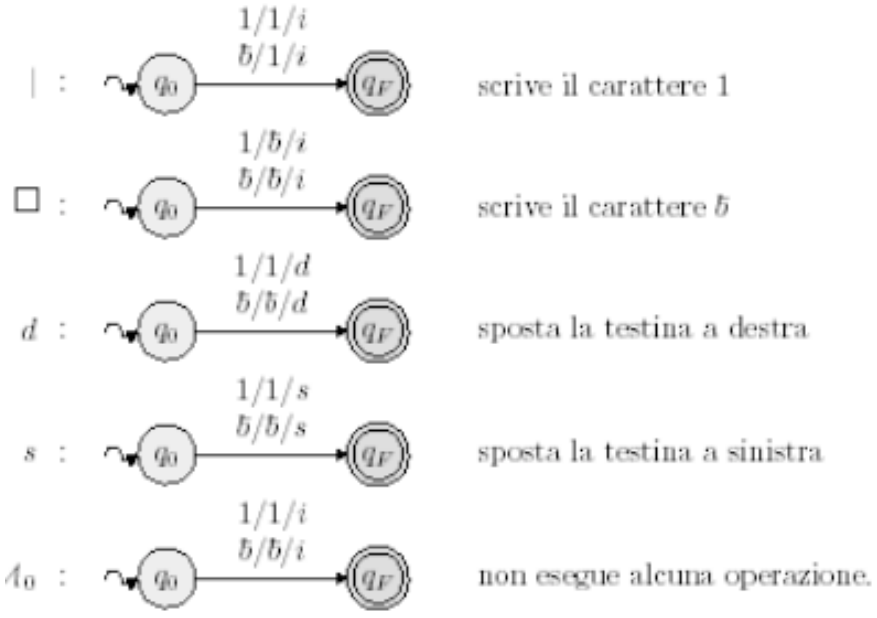
E' possibile rappresentare le MDT attraverso concatenazione di diramazioni di macchine di turing elementari.

Tale processo consiste nel far subentrare, alla terminazione del programma di una MDT, una nuova macchina che prosegue l'algoritmo.

- La MDT composta ha un alfabeto Γ equivalente all'unione dell'alfabeto di ogni singola macchina interna.
- L'insieme degli stati della macchina composta è pari all'unione degli stati delle macchine che la compongono.
- L'insieme degli stati finali della macchina composta è pari all'unione degli stati finali delle macchine che terminano la computazione della macchina composta.
- L'insieme delle funzioni di transizione della macchina composta sarà l'unione delle regole di transizione di tutte le regole di ciascuna macchina, più le regole di transizione tra le macchine che compongono la composta.

Definiamo tramite concatenazione, una tipologia di macchine di turing elemntari che lavorano solo sul simbolo 1 più il ø.

Definiamo le MDT elementari chiamate I, □, d, s, M_0 definite sull'alfabeto Γ = {1} nel modo seguente:



Possiamo dimostrare che è possibile rappresentare ogni macchina di turing come concatenazione di queste macchine elementari.

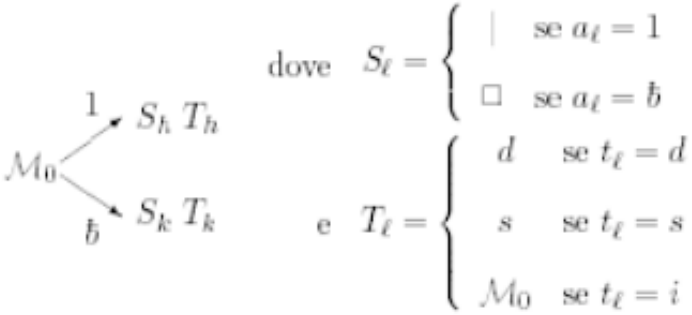
La dimostrazione del fatto che ogni macchina di turing può essere descritta da una macchina elementare è ovvia, in quanto componendo le macchine di turing elementari, ottengo sempre una nuova macchina di turing.

Dimostrare invece che una macchina di turing qualsiasi può essere scomposta in MDT elementari, è necessario osservare che ad ogni MDT possiamo associare un piccolo diagramma che utilizza le macchine elementari. Infatti considerando q_i un generico stato nella macchina, si osserva che lo stato può appartenere a due regole di transizione:

- $\sigma(q_i,1) = (q_h,a_h,t_h)$
- $\sigma(q_i,ø) = (q_k,a_k,t_k)$

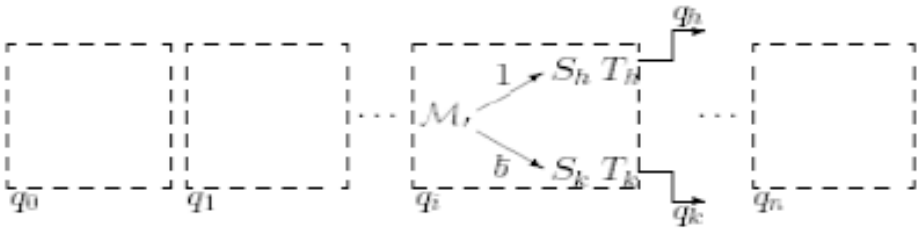
con a_k e $a_h \in \{1,ø\}$ e t_h e $t_k \in \{d,s,i\}$

A tale stato possiamo far corrispondere il seguente diagramma:



con l=h oppure l=k, a seconda del caso.

Il diagramma dovrà poi collegarsi ai diagrammi corrispondenti agli stati q_h e q_k, come nello schema seguente:



La macchina di Turing universale

- La macchina universale è in grado di simulare ogni altra macchina di Turing.
- Può avere sul suo nastro o un dato o un'altra macchina di Turing
- E' definita come U = <Γ, ø,Q',σ', q'_0,F'> .
- E' detta universale se data un'altra macchina di turing e il suo nastro, ottiene lo stesso risultato della macchina simulata.
- Si dice macchina di Turing universale se calcola una funzione u: $\Gamma^{*(n+1)} \rightarrow \Gamma^*$

- Si supponga di avere una macchina con:
  - Nastro semi infinito
  - Alfabeto composto da {1,ø}
- Si prova che è possibile fornire una descrizione linearizzata di ogni macchina di turing del tipo descritto utilizzando solo 3 tipi di macchine di Turing elementari e composizione per diramazione sul simbolo 1.
- Si mostra, infine, come questa descrizione linearizzata può essere rappresentata essa stessa mediante un'alfabeto {1,ø} e può essere definita una macchina che, dato tale input, simula il comportamento della macchina descritta.

Passaggi

1. Componiamo alcune delle MT elementari, così da usarne solo 3, semplificando le nostre operazioni:

- Macchina di Turing j: legge il carattere osservato e sposta la testina a destra
- Macchina di Turing s: che sposta la testina a sinistra
- Macchina di Turing M_0: termina il programma

2. Si definisce che tutti i salti condizionati vengono effettuati solo e soltanto sul simbolo 1 (mai sul blank ø)
3. Semplifichiamo M_0 assumendo che tale macchina sia posta solo alla fine della macchina di Turing (più avanti si definisce come si identificherà quindi la fine del programma

A questo punto abbiamo:

1. Macchina j e macchina s.

2. Codifichiamo tali macchine attraverso stringhe di lunghezza finita composte solo dal simbolo {1}

Codifica

- j = "11"
- s = "1"
- un qualsiasi salto condizionato: = "1^{3+n}"
  dove n è la cella a cui si vuole saltare. (Per es: n=2 = 11111)

- Al termine del programma ci sono 3 simboli 1 “111”, che rappresentano la fine del programma sostituendo così M_0

Quindi:

Tale macchina di Turing così codificata più i suoi input, viene data in pasto alla macchina universale che restituisce come output lo stesso risultato della macchina di Turing di cui si è effettuata la codifica.

All’interno di questa macchina universale, per tenere traccia dell’istruzione che si sta elaborando e del dato su cui si sta lavorando, si utilizzano due caratteri speciali: 1’, ø’

Problema dell’alt (*)

da rivedere sugli appunti di carta