UniCT – DMI

Fondamenti d'Informatica

ANNO ACCADEMICO

2019/20

Autori:

Alessio Tudisco

Sommario

Fondamenti D'Informatica

Logica

Definizione di Sistema Formale e di Linguaggio Formale

Un sistema formale D è dato da:

- Un insieme numerabile S, detto alfabeto;
- Un insieme decidibile $W \subseteq S^*$, detto insieme *delle formule ben formate* (FBF);
- Un insieme $Ax \subseteq W$, detto insieme degli *assiomi del sistema formale*. Se Ax è decidibile, diremo che il sistema formale è ricorsivamente assiomatizzato;
- Un insieme $R = \{R_i\}_{i \in I}, con\ R_i \subseteq W^{n_i}\ con\ I\ ed\ n_i \geq 2\ finiti$, un insieme di regole che ci permettono di derivare tutti e soli i giudizi validi;

La coppia < S, W > è detta linguaggio formale.

Osservazioni:

- Una FBF di un sistema formale è una stringa di simboli che rappresenta un'espressione sintatticamente corretta, definita mediante le regole del sistema formale stesso;
- Una regola di inferenza è uno schema formale che, partendo da una o più premesse, riesce ad arrivare ad una conseguenza/conclusione;
- Un assioma è un principio che viene assunto come vero perché ritenuto evidente o perché fornisce il punto di partenza di un quadro teorico di riferimento;

Definizione di 'Definizione Esplicita'

Dicesi definizione esplicita la definizione di un termine che viene aggiunto all'alfabeto del linguaggio per significare un'espressione.

Definizione di D-derivazione e derivabilità

Dato un insieme M di FBF nel sistema formale D, una *D-derivazione* (prova, dimostrazione) a partire da M è una successione finita di FBF $a_1, \dots, a_n$ di D tale che, per ogni $i = 1, \dots, n$ si abbia:

- $a_i \in Ax$ oppure
- $a_i \in M$ oppure
- $\left(a_{h_1}, \dots, a_{h_{n_j}}\right) \in R_j\ per\ qualche\ j \in I$

Una formula α è derivabile nel sistema formale D a partire da un insieme di ipotesi M se e solo se esiste una D-derivazione a partire da M la cui ultima FBF è α. Scriveremo allora $M \vdash_D \alpha$ e leggeremo: M deriva α nel sistema formale D.

Se M è vuoto scriveremo $\vdash_D \alpha$ e leggeremo: α è un teorema in D se e solo se non vale $M \vdash_D \alpha$.

Definizione di Regola Derivabile e Ammissibile

Sia R l'insieme delle regole di un sistema formale D, una regola $r = \frac{a_1, \dots, a_k}{a_{k+1}}$ è detta:

- *derivabile in D* se e solo se per tutte le FBF $a_1, \dots, a_k$ che soddisfano r si ha: $a_1, \dots, a_k \vdash_D a_{k+1}$;
- *ammissibile (o eliminabile) in D* se e solo se da $M \vdash_{D \cup \{r\}} \alpha$ segue $M \vdash_D \alpha$, dove $D \cup \{r\}$ denota il sistema formale ottenuto da D con l'aggiunta della regola r;

Osservazioni

- *Ogni regola derivabile è ammissibile, ma non l'inverso*: data una D-derivazione $M \vdash_{D \cup \{r\}} \alpha$ posso sostituire ogni applicazione della regola r con la sua corrispondente *D-derivazione*, che esiste in D poiché è r è derivabile;
- Se $M \vdash_D \alpha$ allora esiste $N \subseteq M\ con\ N\ finito$, per il quale si ha $N \vdash_D \alpha$, le dimostrazioni sono esponibili in tempo finito: $M \vdash_D \alpha$ se e solo se $\exists\ (derivazione)\ a_1, \dots, a_n \equiv a$, solo un numero minore o uguale a n (numero finito), di a_i appartiene a M, perciò $N = M \cap \{a_1, \dots, a_n\}$;

- Se $M \mid\!-_D \alpha_1, \ldots, M \mid\!-_D \alpha_n$ e $\{a_1, \ldots, a_n\} \mid\!-_D \beta$, allora si avrà $M \mid\!-_D \beta$

Definizioni di Correttezza, Completezza, Consistenza e Inconsistenza di un Sistema Formale

Un *sistema formale D* si dice:

- *D è corretto* rispetto a una semantica: quando i suoi assiomi sono tutti validi e quando le regole sono tali da garantire che se le premesse dei giudizi sono validi allora la conclusione sarà anch'essa valida, ovvero tutti i teoremi derivati dal sistema sono validi;
- *D è completo* rispetto a una semantica: quando tutti i giudizi validi sono derivabili nel sistema formale;
- *D è consistente* quando esiste almeno un FBF α tale $\not\vdash_D \alpha$ (non è derivabile), altrimenti D è *inconsistente*;

Definizione di Insieme delle Conseguenze

Sia M un insieme finito di FBF di un sistema formale D, si *definisce insieme delle conseguenze di M in D* l'insieme $con_D(M) = \{a \in W \; tali \; che \; M \mid\!-_D a\}$

Definizione di Insieme Finito (di FBF) Consistente e Inconsistente

Sia M un insieme finito di FBF di un sistema formale D, diremo che:

- M *è consistente rispetto a D* se e solo se esiste $a \in W \; tale \; che \; M \not\vdash_D \alpha$ (non è derivabile), ovvero si ha che $con_D(M) \neq W$;
- M è detto *inconsistente rispetto a D* se e solo se M non è consistente;

Definizione di Teoria e Teoria Pura

Un insieme M di FBF di un *sistema formale D è detto teoria* se e solo se M è chiuso rispetto alla relazione di D-derivazione, ovvero se $con_D(M) = M$ quindi si ha che se $M \mid\!-_D a \; allora \; a \in M$.

La *teoria pura* di *un sistema formale D* è l'insieme $con_D(\emptyset) = con_D(Ax)$.

Definizione di Sistema Formale P_0(Calcolo Proposizionale)

Chiamiamo:

- *Proposizioni* delle espressioni elementari che possono possedere un valore di verità;
- *Variabili proposizionali* delle variabili che stanno per proposizioni (solitamente denominate p, q, r …);
  - ❖ Un assegnamento proposizionale è una funzione che associa ad ogni variabile proposizionale un valore di verità (0,1);
- *Tautologia* una FBF il cui valore di verità è sempre 1;

Il calcolo proposizionale è un sistema formale P_0 i cui teoremi sono tutte e sole le tautologie.

Il sistema formale P_0 è il seguente sistema formale:

- *S*: è formato dall'unione di un insieme numerabile di *variabili proposizionali*, l'insieme dei *connettivi logici* $\neg(NOT), \; \wedge (AND), \; \vee (OR), \rightarrow (IMPLICA), \leftrightarrow (DOPPIA \; IMPLICAZIONE)$ e l'insieme dei due simboli ausiliari (*e*);
- W: l'insieme delle FBF così definito:
  - ❖ Ogni variabile proposizionale è una FBF;
  - ❖ Se α e β sono FBD allora lo sono anche $(\alpha \rightarrow \beta)$ *e* $(\neg\alpha)$;
    - $(\alpha \vee \beta) = \big((\neg a) \rightarrow \beta\big) \mid (\alpha \wedge \beta) = \neg\big((\neg a) \vee (\neg\beta)\big) \mid (\alpha \leftrightarrow \beta) = (\alpha \rightarrow \beta) \wedge (\beta \rightarrow \alpha)$
  - ❖ Nient'altro è una FBF;
- Ax: è definito come:
  - ❖ $\alpha \rightarrow (\beta \rightarrow \alpha)$ [AK]
  - ❖ $(\alpha \rightarrow (\beta \rightarrow \gamma)) \rightarrow ((\alpha \rightarrow \beta) \rightarrow (\alpha \rightarrow \gamma))$ [AS]
  - ❖ $(\neg\beta \rightarrow \neg\alpha) \rightarrow ((\neg\beta \rightarrow \alpha) \rightarrow \beta)$ [A¬]
- $R = \{MP\}$ dove MP (*modus ponens*) è la regola $\frac{a \; a \rightarrow \beta}{\beta}$

Il modus pones è una semplice e valida regola di inferenza che afferma: *se p implica q è una proposizione vera, e anche la premesse p è vera, allora la conseguenza q è vera.*

Dimostrazione che: $|-_{P_0} a \to a$ è teorema:

Applichiamo l'assioma *AS* ponendo $\beta = a \to a \; e \; c = a$

1. $(a \to ((a \to a) \to a)) \to ((a \to (a \to a) \to (a \to a))$
   Applichiamo l'assioma AK ponendo $\beta = a \to a$
2. $(a \to ((a \to a) \to a)$
   Applichiamo il modus pones usando come proposizione la 1) e come premessa la 2)
3. $(a \to (a \to a) \to (a \to a)$
   Applichiamo l'assioma AK con $\beta = a$
4. $a \to (a \to a)$
   Applichiamo il modus pones (3,4)
5. $a \to a$

Definizione di Teorema di deduzione (Deduzione di Herbrand)

Dato un insieme di FBF M in P_0 si ha che:

$$M|-_{P_0} \alpha \to \beta \leftrightarrow M, \alpha|-_{P_0} \beta$$

Semantica di P_0: Definizione di Assegnazione B *e e valutazione* $\overline{B}$

Un *assegnamento proposizionale B* è una funzione:

$$B : variabili\ proposizionali \to \{0,1\}$$

Un *assegnamento proporzionale B* è esteso per induzione ad una valutazione $\overline{B}$ del linguaggio di P_0 nel modo seguente:

- $\overline{B}(p) = B(p) \; \forall \; p$
- $\overline{B}(a \to \beta) = \{0 \leftrightarrow \overline{B}(a) = 1 \; e \; \overline{B}(\beta) = 0; 1 \; altrimenti\}$, cioè se vero implica falso;
- $\overline{B}(\neg a) = 1 - \overline{B}(a)$

Negazione ($\neg = NOT$)	
P	¬P
0	1
1	0

Congiunzione ($\wedge = AND$)		
P	Q	P∧Q
0	0	0
0	1	0
1	0	0
1	1	1

Disgiunzione (V= OR)		
P	Q	PVQ
0	0	0
0	1	1
1	0	1
1	1	1

Implicazione (→)		
P	Q	P→Q
0	0	1
1	0	0
0	1	1
1	1	1

priorità più alta	¬
.	∧
.	V
priorità più bassa	→

Semantica di P_0: Tipologie di FBF

Una FBF a di P_0 è detta:

- *Tautologia:* se e solo se per ogni $assegnamento\ proposizionale\ B\ si\ ha\ \overline{B}(a) = 1$, cioè se è sempre vera;
- *Soddisfacibile:* se e solo se esiste un $assegnamento\ proposizionale\ B\ tale\ che\ \overline{B}(a) = 1$, cioe se è vera almeno in un caso;
- *Contradditoria:* se e solo se non è *soddisfacibile*, cioè se non è mai vera;

Un insieme M di FBF di P_0 è detto soddisfacibile se e solo se esiste un assegnamento proposizionale B tale che:

$$\overline{B}(\beta) = 1 \; \forall \beta \in M$$

- Una FBF a di P_0 è detta è detta *tautologia* se e solo se $\neg a$ è contradditorio;
- Se $a \; e \; a \to \beta$ sono *tautologie,* allora anche β è una tautologia;

- Data FBF a di P_0 è decidibile il problema se a è una tautologia, si crea una tabella di verità per a e si controlla se ogni riga ha $\overline{B}(a) = 1$;

Sia a una FBF di P_0, $\overline{B}(a)$ dipende solo dal valore assegnato da B alle *variabili proposizionali* presente in a.

Definizioni di Consistenza-Inconsistenza delle FBF di P_0

Teorema: Un insieme M di FBF di P_0 è inconsistente, cioè $con_{P_0}(M) = W$, se e solo se esiste una FBF $a\ tale\ che\ M|-_{P_0} a\ e\ M|-_{P_0} \neg a$.

Sia M un insieme di FBF *soddisfacibile* di P_0, allora M è consistente, e viceversa.

Sia M un insieme di FBF consistente di P_0 e sia $a\ tale\ che\ M \not|-_{P_0} a$ (M non deriva a) allora $M \cup \{\neg a\}$ è consistente.

Se $M \cup \{a\}$ è inconsistente allora $M|-_{P_0} \neg a$.

Semantica di P_0: Conseguenza Tautologica

Data una $FBF\ a$ di P_0 e un insieme M di $FBF\ di\ P_0$, si dice che a è conseguenza tautologica di M se e solo se per ogni *assegnamento proposizionale B* si ha che: se per $\forall \beta \in M$ vale che $\overline{B}(\beta) = 1$ allora $\overline{B}(a) = 1$.

Indichiamo che a è conseguenza tautologia di M con $M \vDash a$.

-B(b) = 1 allora -B(a) = 1. Quinsi se per ogni assegnamento che facciamo a b appartenente a V che ci da verità, anche "a" sarà vera.

Definizioni di Correttezza e Completezza del Sistema Formale P_0

Teorema di Correttezza: Siano M un insieme di $FBF\ di\ P_0$ e $a\ una\ FBF\ di\ P_0$, si ha che:

$$se\ M|-_{P_0} a\ allora\ M \vDash a$$

Perciò, P_0 è consistente: se fosse inconsistente, esisterebbe una $FBF\ a$ tale che $|-_{P_0} a\ e\ |-_{P_0} \neg a$ e per il teorema di correttezza sia $a\ che\ \neg a$ sarebbero tautologie, il che è assurdo per la definizione di tautologia.

Teorema di Completezza: Siano M un insieme di $FBF\ di\ P_0$ e $a\ una\ FBF\ di\ P_0$, si ha che:

$$se\ M \vDash a\ allora\ \ M|-_{P_0} a$$

Possiamo quindi dire che: Siano M un insieme di $FBF\ di\ P_0$ e $a\ una\ FBF\ di\ P_0$

$$M \vDash a \leftrightarrow M|-_{P_0} a$$

Data FBF a di P_0 è decidibile il problema se a è un teorema di P_0.

Deduzione Naturale in Logica Proposizionale

Il sistema formale per il calcolo proposizionale descritto sopra è un sistema *assiomatico*, detto anche alla Hilbert, in quanto l'insieme delle regole di inferenza è composto da una sola regola: il modus ponens.

Esistono molti altri sistemi formali, tra cui quello della *deduzione naturale* in cui:

- L'insieme degli assiomi Ax è vuoto: $Ax = \emptyset$;
- Ci sono solo regole d'inferenza;

Rispetto ai sistemi assiomatici cambia solo un poco la nozione di *derivazione*:

In un sistema di deduzione naturale si ha che: $M|-\ \ P$ (*P è derivabile dall'insieme di premesse M)* quando è possibile costruire un albero di derivazione di P da *premesse di M*.

Un sistema assiomatico e la deduzione naturale sono equivalenti dal punto di vista della derivabilità, perché partendo da un insieme M di ipotesi, l'insieme delle conseguenze di M è identico in entrambi.

Logica dei Predicati: Segnatura

Si definisce *segnatura* una coppia formata da un insieme di simboli di funzione e un insieme i simboli di predicato.

Aggiungiamo un esponente intero ad ogni simbolo di funzione e predicato quando vogliamo specificare l'arietà, ovvero quanto argomenti richiede la funzione.

Esempio: $\Sigma_{PA} = \{\{0^0, s^1\}, \{<^2\}\}$

- Un simbolo di costante è un simbolo di funzione $0 - aria$;
- L'insieme dei termini su una segnatura Σ è indicata con $Term_\Sigma$ ed è definito:
  - $x \in Term_\Sigma \; \forall x \; simbolo \; di \; variabile \; individuale$;
  - $se \; t_1, \dots, t_n \in Term_\Sigma \; e \; f^n$ è $un \; simbolo \; di \; funzione \; n_{aria} \; di \; \Sigma, allora \; f^n(t_1, \dots, t_n) \in Term_\Sigma$;
  - $Nient'altro$ è $un \; termine \; della \; segnatura \; \Sigma$;

Logica dei Predicati: Linguaggio dei Predicati

Sia Σ una segnatura con un numero al più numerabile di simboli di predicato e simboli di funzione; il linguaggio dei predicati su Σ, L_Σ, è il seguente linguaggio formale:

- alfabeto S:
  - un insieme di variabili individuali (x; y; x_0; y_0; …) di cardinalità arbitrariamente grande;
  - i simboli di funzione ($indicati \; con \; f, g, h$ …) e di predicato ($indicati \; con \; P, Q, R$ …) in Σ;
  - i connettivi proposizionali ¬ e →;
  - un simbolo di quantificazione universale: ∀;
  - due simboli ausiliari: (e);
  - eventualmente un simbolo di uguaglianza =, e in caso esso sia presente il linguaggio è detto $linguaggio \; di \; uguaglianza$;

  Tutti questi insiemi devono essere decidibili e distinti

- insieme delle FBF indicato con fbf_Σ, definito come:
  - se $t_1, \dots, t_n \in Term_\Sigma$ e P^n è un simbolo di predicato n-aria di Σ, allora $P^n(t_1, \dots, t_n) \in fbf_\Sigma$ (fbf atomiche);
  - se $\alpha, \beta \in fbf_\Sigma$ allora $\alpha \to \beta \in fbf_\Sigma \; e \; \neg\alpha \in fbf_\Sigma$;
  - se $\alpha \in fbf_\Sigma \; e \; x$ è una variabile individuale, allora $(\forall x \; \alpha) \in fbf_\Sigma$;
  - se il linguaggio è un *linguaggio con uguaglianza* e $t_1 e \; t_2 \in Term_\Sigma, allora \; t_1 = t_2 \in fbf_\Sigma$;
  - *nient'altro è in* fbf_Σ;

Logica dei Predicati: Struttura i A_Σ

Sia Σ una segnatura, si definisce struttura A_Σ per Σ tripla $A_\Sigma = (A, F, P)$ dove:

- A è un insieme detto *supporto di* A_Σ, *spesso indicato con* $|A_\Sigma|$;
- F è una famiglia di funzioni tale che $\forall f^n \in \Sigma \;\; \exists \; \widetilde{f^n} \in$ F tale che $\widetilde{f^n}: A^N \to A$;
- P è una famiglia di relazioni tale che $\forall P^n \in \Sigma \;\; \exists \; \widetilde{P^n} \in P \; tale \; che \; \widetilde{P^n}: A^N \to \{0,1\}$;

Logica dei Predicati: Ambiente

Sia Σ una segnatura, L_Σ il linguaggio definito in essa, A_Σ la struttura definita in essa, un ambiente $\rho_{(L_\Sigma, A_\Sigma)}$ è un'applicazione: $\rho: variabile_{individuale} \to |A_\Sigma|$;

Si definisce l'insieme degli ambienti per un linguaggio L_Σ e una struttura A_Σ:

$$env_{L_\Sigma, A_\Sigma} = \{\rho: variabile_{individuale} \to |A_\Sigma|\}$$

Logica dei Predicati: Interpretazione per Termini

Un'interpretazione per termini su una struttura A_Σ è una funzione:

$$[\![\_]\!]^{A_\Sigma}_{\_} : \mathrm{Term}_\Sigma \times ENV^{A_\Sigma} \to |A_\Sigma| \quad tale \; che:$$

- $[\![x]\!]_\rho = \rho(x) \; \forall x \in variabili_{individuali} \; di \; L_\Sigma$;

- $[\![f^n(t_1,\dots,t_n)]\!]_\rho = \left(f^n([\![t_1]\!]_\rho,\dots,[\![t_n]\!]_\rho\right) \forall\ (t_1,\dots,t_n) \in Term_\Sigma$;

Logica dei Predicati: Interpretazione per FBF

Un'interpretazione per FBF su una struttura A_Σ è una funzione:

$$[\![\_]\!]^{A_\Sigma}_{\_} : \text{fbf}_\Sigma \times ENV^{A_\Sigma} \to \{0,1\} \quad tale\ che:$$

- $[\![P^n(t_1,\dots,t_n)]\!]_\rho = P^n([\![t_1]\!]_\rho,\dots,[\![t_n]\!]_\rho)$;
- $[\![a \to \beta]\!]_\rho = 1 \leftrightarrow [\![a]\!]_\rho = 1\ e\ [\![\beta]\!]_\rho = 1$;
- $[\![\neg a]\!]_\rho = 1 \leftrightarrow [\![a]\!]_\rho = 0$;
- $[\![\forall x.a]\!]_\rho = 1 \leftrightarrow \forall a \in |A_\Sigma|\ si\ ha\ [\![a]\!]_{\rho^a_x} = 1\ dove\ \rho^a_x \in ENV^{A_\Sigma}$;
- Se il linguaggio è un linguaggio con uguaglianza, $[\![t_1 = t_2]\!]_\rho = 1 \leftrightarrow [\![t_1]\!]_\rho\ e\ [\![t_2]\!]_\rho\ sono\ lo\ stesso\ elemento\ in\ A_\Sigma$

Logica dei Predicati: Tipologie di FBF

Sia $a\ in\ fbf_\Sigma\ e\ A\_\Sigma$ una struttura, diremo che:

1. a è soddisfacibile in $A_\Sigma \leftrightarrow \exists\ \rho \in ENV^{(A_\Sigma)}\ tale\ che\ [\![a]\!]_\rho = 1$;
2. a è valida in $A_\Sigma \leftrightarrow \forall\ \rho \in ENV^{(A_\Sigma)}\ si\ ha\ che\ [\![a]\!]_\rho = 1$; scriveremo che $A_\Sigma \vDash a$
   L'insieme dele teorie di A_Σ: $Th(A_\Sigma) = \{a\ tale\ che\ A_\Sigma \vDash a\}$
3. a è soddisfacibile $\leftrightarrow \exists\ A_\Sigma\ che\ soddisfa\ a$:
4. a è valida $\leftrightarrow \forall\ A_\Sigma\ si\ ha\ che\ A_\Sigma \vDash a$;
5. a è contraddittoria $\leftrightarrow a\ non$ è $soddisfacibile$;

Logica dei Predicati: Modello

Siano M un insieme di FBF $M \subseteq fbf_\Sigma$ e A_Σ una struttura, A_Σ è detta modello di M di $A_\Sigma \leftrightarrow \forall a \in M\ si\ ha\ che\ a$ è $vera\ in\ M$ ($A_\Sigma \vDash$ a). Allora scriveremo che $A_\Sigma \vDash$ M.

Logica dei Predicati: Testimone dell'esistenziale

Denotato: $(\exists x.a) \equiv \neg(\forall x.\neg a)$

Sia A_Σ una struttura; $A_\Sigma \vDash \exists x.a \leftrightarrow \forall \rho \in ENV^{A_\Sigma}\ \exists\ a \in |A_\Sigma|\ tale\ che\ [\![a]\!]_{\rho^a_x} = 1$.

Diremo che a è detto testimone dell'esistenziale.

Logica dei Predicati: Conseguenza e Validità di una FBF

Siano M un insieme di FBF $M \subseteq fbf_\Sigma$ e $a \in fbf_\Sigma$ diremo che:

- a è conseguenza logica di M $\leftrightarrow \forall A_\Sigma\ e\ \forall \rho \in\ ENV^{A_\Sigma}\ si\ ha\ che\ [\![M]\!]_\rho = 1\ e\ [\![a]\!]_\rho = 1, scriveremo\ che\ M \vDash a$;
- a è valida in M $\leftrightarrow \forall A_\Sigma$ dal fatto che $\forall \rho \in\ ENV^{A_\Sigma}\ si\ ha\ [\![M]\!]_\rho = 1\ e\ \forall\ \rho \in\ ENV^{A_\Sigma}\ si\ ha\ [\![a]\!]_\rho = 1$

Siano M un insieme di FBF $M \subseteq fbf_\Sigma$ e $a \in fbf_\Sigma$: se $M \vDash a\ allora\ a$ è $valida\ in\ M$.

Dimostrazione: Se per assurdo a non fosse valida in M, allora esisterebbe una struttura A_Σ per cui $\forall p \in ENV^{A_\Sigma}$ $[\![M]\!]_\rho = 1$ e per almeno un $p' \in ENV^{A_\Sigma}\ si\ ha\ che\ [\![a]\!]_{\rho'} = 0$. Quindi a non potrebbe essere conseguenza logica di M, assurdo.

L'insieme delle FBF valide $T = \{a\ tale\ che \vDash a\}$ è ricorsivamente enumerabile:

- Grazie ai teoremi di correttezza e completezza, un algoritmo ovvio è quello di cominciare dagli assiomi base e di applicare in tutti i modi possibili la regola del modus ponens, quindi replicare il procedimento allungando gli assiomi.

Logica dei Predicati: Teorema di Correttezza, Completezza e Compattezza

Teorema della correttezza: Siano M un insieme di FBF $M \subseteq fbf_\Sigma$ e $a \in fbf_\Sigma$ si ha che:

$$se\ M|-_{PL}a\ allora\ M \vDash a$$

Teorema della completezza: Siano M un insieme di FBF $M \subseteq fbf_{\Sigma}$ e $a \in fbf_{\Sigma}$ si ha che:

$$se\ M \vDash a\ allora\ M|-_{PL}a$$

Teorema della compatezza: Sia M un insieme di FBF $M \subseteq fbf_{\Sigma}, M$ è $soddisfacibile \leftrightarrow \forall \Delta \subseteq M\ di\ cardinalità\ finita$ è $soffisfacibile$.

Aritmetica e teoria dei gruppi (Peano)

Per poter rappresentare tutti i numeri naturali sono sufficienti due simboli:

- una costante che indichi lo zero;
- una funzione unaria che rappresenti il successore di un numero.

Per poter fare dei conti utilizziamo i simboli di funzione per la somma e per la moltiplicazione e il predicato di uguaglianza. Quindi: $\{0, s, +, *, =\}$

Per esempio, la somma tra 1 e 2 sarà: $| - \ s(0) + s(s(0)) \ = \ s(s(s(0)))$

Ciò non è però possibile poiché il teorema di correttezza ci dice che se possiamo dimostrare questo, allora è una conseguenza tautologica, ma così non è, perché in altre strutture, dove possiamo dare significati diversi ai simboli che utilizziamo, la stessa espressione può non essere vera.

Per risolvere il problema dobbiamo porre dei vincoli al comportamento dei simboli base, tramite delle ipotesi, l'insieme di formule ben formate PA, che renderanno l'espressione una conseguenza tautologica di PA, cioè l'espressione è vera in tutte le strutture che sono modelli di PA:

1. ∀x(¬0 = s(x)). Per ogni x, not 0 è uguale al successore di x.
2. ∀xy(s(x) = s(y) → x = y). Per ogni x e per ogni y, se il successore di x è uguale al successore di y, allora x e y sono uguali.
3. ∀x(x + 0 = x). Per ogni x, x più 0 è uguale a x.
4. ∀xy(x + s(y) = s(x + y)). Per ogni x e per ogni y, x più il successore di y è uguale al successore di x + y.
5. ∀x(x ∗ 0 = 0). Per ogni x, x moltiplicato con 0 è uguale a 0.
6. ∀xy(x ∗ s(y) = x + (x ∗ y)). Per ogni x e per ogni y, x moltiplicato per il successore di y è uguale a x sommato al prodotto tra x e y.
7. P(0) ∧ ∀x(P(x) → P(s(x))) → ∀xP(x).

Lambda Calcolo

Introduzione

Un modello computazionale è una specifica formalizzazione del concetto di computazione, ovvero il processo in cui trasformiamo un'informazione da una forma implicita a una esplicita. Ci sono vari modelli computazionali: *Macchine di Turing, Macchina a Registro Illimitato, Sistemi di Post e lambda-calcolo.*

I vari linguaggi di programmazione che usiamo oggigiorno sono, direttamente o indirettamente, ispirati a uno o più modelli computazionali. A seconda del modello computazionale a cui si ispirano possiamo classificarli come:

- *Linguaggi di programmazione imperativi* (C, C++): sono basati sui modelli come quello delle *Macchine di Turing e le Macchina a Registro Illimitato,* gli aspetti fondamentali sono:
  - Memoria;
  - Variabile, qualcosa che può essere letto e modificato (cella di memoria);
  - Assegnamento;

    - ❖ Istruzione, comando;
    - ❖ Iterazione, ciclo;
- *Linguaggi di programmazione funzionali (Haskell)*: sono basati sul modello del lambda-calcolo e gli aspetti fondamentali sono:
    - ❖ Espressione;
    - ❖ Ricorsione, diverso dal concetto di iterazione;
    - ❖ Variabile, intesa nel senso matematico: denota un'entità sconosciuta o un'astrazione di un valore concreto;
    - ❖ Valutazione, diverso dal concetto di esecuzione in quanto non viene eseguita nessuna alterazione;
- Linguaggi di programmazione logici: si basano su alcune proprietà di sottoinsiemi della logica del primo ordine, computare significa cercare una deduzione in forma logica;

Un programma in un linguaggio funzionale consiste in un insieme di definizioni di funzioni e in una espressione, normalmente formata tramite le funzioni prima definite. L'interprete del linguaggio valuta questa espressione tramite un numero discreto di passi di computazioni e la fa passare in una forma esplicita. Quindi, durante questo tipo di computazioni, nulla viene creato: la rappresentazione dell'informazione viene trasformata.

La computazione nel linguaggio computazionale è suddivisa in tre passi:

- Nel primo passo sostituiamo un nome con l'espressione ad esso associata;
- Nel secondo sostituiamo il parametro attuale al posto del parametro formale;
- Nel terzo vi è il calcolo di una funzione predefinita a cui abbiamo applicato un valore esplicito;

Spesso, quando valutiamo un'espressione, ci sono più di una sotto-espressione che possiamo valutare e scegliere una piuttosto che un'altra ci porta ad avere diversi modi di valutare una stessa espressione. Una *strategia di valutazione* è una politica che ci permette di decidere quale sotto-espressione valutare. Alcune strategie di valutazione possono portarci ad un loop di valutazioni senza fine ma, grazie al Teorema di Confluenza del Lambda-calcolo, tutti i percorsi di valutazione finiti ci portano allo stesso risultato. In un'espressione il valore di ogni sua sotto-espressione non dipende ne dal modo in cui l'abbiamo valutata fino a quel momento ne dal valore delle altre, perché la stessa espressione denota sempre lo stesso valore (trasparenza referenziale).

Lambda Calcolo: Termini

Il lambda calcolo è un linguaggio formale le cui espressioni sono *chiamate lambda termini*. Chiamiamo lambda termine qualunque stringa generata dalla seguente grammatica:

$$T ::= Id|\lambda Id.T|(T\ T)$$

dove *Id* appartiene a un insieme numerabile e infinito di variabili.

Un lambda termine può dunque essere, rispettivamente alla grammatica, un nome di variabile, l'astrazione di un termine rispetto ad una variabile (funziona anonima) o l'applicazione di un termine come argomento (o parametro) di un altro.

Convenzioni per le semplificazioni delle parentesi:

- M N invece di (M N)
- M N P invece di (M N)P
- $\lambda x.\ M\ N$ invece di $\lambda x.\ (M\ N)$
- $\lambda xyz.\ M$ invece di $\lambda x.\ \lambda y.\ \lambda z.$ M

Lambda Calcolo: Regole di Riscrittura

Definiamo come i lambda termini possono essere riscritti.

- Variabili: dato un generico lambda termine T:
    - ❖ Chiamiamo $Var(T)$ l'insieme che contiene tutte le variabili menzionate in T, raccoglie tutti i nomi delle variabili presenti in T;
    - ❖ Chiamiamo $Libere(T)$ l'insieme che contiene le sole variabili libere di T definito come:
        - $Libere(x) = \{x\}$;

- $Libere(\lambda x.M) = Libere(M) - \{x\}$;
- $Libere(M\ N) = Libere(M) \cup Libere(N)$;

❖ Chiamiamo $Legate(T)$ l'insieme con tutte le variabili legate, ovvero che non sono libere:

- $Legate(T) = Var(T) - Libere(T)$;

Esempio:
$Var(\lambda z.\lambda x.(xy)) = \{z, x, y\};\ Libere(\lambda z.\lambda x.(xy)) = \{y\};\ Legate(\lambda z.\lambda x.(xy)) = \{x, z\}$

❖ Sostituzione: Chiamiamo sostituzione il rimpiazzo di tutte le occorrenze di un sotto-termine con un altro, all'interno di un terzo termine che rappresenta il contesto della sostituzione stessa. Indichiamo con:
$T1\ [T2/T3\ ]$ la sostituzione del termine $T2$ al posto di $T3$ all'interno del termine-contesto $T1$.

$$y\ [L/x] \equiv \begin{cases} L & \text{if } x=y \\ y & \text{if } x\neq y \end{cases} \qquad PQ\ [L/x] = P[L/x]\ Q[L/x] \qquad \lambda y.P[L/x] \equiv \begin{cases} \lambda y.P & \text{if } x=y \\ \lambda y.(P\ [L\ /x]) & \text{if } x\neq y \end{cases}$$

La sostituzione avviene solo nelle variabili libere! Non in quelle legate (ovvero con λ). Inoltre una sostituzione non deve rendere legate le variabili che prima erano libere;

❖ L'alfa-conversione: si applica ai termini che sono astrazioni. Data un'astrazione, possiamo riscriverla sostituendo la variabile astratta (x) con un'altra (y), a patto che, nell'intero sotto-termine, al posto di ogni occorrenza della prima, noi andiamo a scrivere la seconda. La regola di Alfa Conversione non si occupa di fare alcuna distinzione fra occorrenze libere o legate delle variabili, dato che l'operazione di sostituzione si occupa già di fare ciò.
Es. $\lambda x.(xy) \Rightarrow \alpha\ \lambda z.(zy)$

❖ β-riduzione: è analoga all'operazione di applicazione di una funzione matematica ai suoi argomenti: il termine sulla destra rappresenta l'argomento (o parametro), mentre il termine più a sinistra rappresenta la funzione stessa che viene applicata. Un termine per essere beta-ridotto deve essere una lambda-astrazione.

$$[REDEX]\ (\lambda x.M)N \Rightarrow \beta\ M\left[\frac{N}{x}\right]$$

Lambda Calcolo: a-equivalenza

Due lambda termini come λx.x e λy.y differiscono solo per il nome di una variabile legata e che quindi fondamentalmente sono la stessa cosa. Due termini così li chiameremo α-equivalenti e li indicheremo come M =α N.

L'alfa-equivalenza è una relazione tra lambda termini che soddisfano tutte le 6 regole:

1. M = M
2. M = N allora N = M
3. M = N e N = P allora M = P
4. M = M' e N = N' allora MN = M'N'
5. M = M' allora λx.M = λx.M'
6. se y non appartiene a M allora λx.M =λy.(M{y/x})

Lambda Calcolo: Forma normale

Diciamo che un termine del lambda calcolo si trova in *forma normale* se esso non è riscrivibile per mezzo della regola di Beta-riduzione.

Nel lambda calcolo, quindi, una funzione calcolabile è rappresentata da una qualche lambda espressione in grado di riscriversi fino a raggiungere un termine in *forma normale*. Viceversa, esistono termini che generano una successione infinita di riscritture senza mai raggiungere una forma normale, che perciò rappresentano funzioni non calcolabili. Un esempio di espressione non calcolabile è la seguente:

$$\lambda y.(yy)\lambda y.(yy) \Rightarrow \beta\ \lambda y.(yy)\lambda y.(yy) \Rightarrow \beta\ \ldots$$

Poiché il termine D = λy.(yy) non fa altro che prendere un termine e scriverne due copie, esso è spesso noto col termine di duplicatore. Logicamente, un termine in forma normale non contiene dei redex.

Lambda Calcolo: Rappresentazione dei Booleani

- T = $\lambda xy.\ x$
- $F = \lambda xy.\ y$

Naturalmente valgono le solite regole dell'and (TT = T, TF = F, FT = F, FF = F), indicato con $\lambda ab.\ abF$.

Si può anche definire il termine if_then_else con $\lambda x.\ x$.

Lambda Calcolo: Numerale di Church

Se f e x sono lambda termini, e $n \geq 0$ è un numero naturale, scriviamo $f^n x$ per indicare $f(f(\dots\ (fx)\ \dots))$ dove f si ripete n volte. Per tutti i numeri naturali n, definiamo il lambda termine $\overline{n}$, chiamato numerale di Church, dato da $\overline{n} = \lambda fx.\ f^n x$.

Lambda Calcolo: Rappresentazione delle funzioni ricorsive

Secondo la Tesi di Church, nel lambda-calcolo possiamo rappresentare qualsiasi funzione che sia calcolabile tramite un algoritmo. Per rappresentare funzioni come il fattoriale, che sono state definite tramite un algoritmo ricorsivo, dobbiamo utilizzare dei lambda termini che rappresentano la nozione stessa di ricorsione, chiamati Operatori di Punto Fisso: Turing e Y. Utilizzando questi operatori si possono rappresentare funzioni definite ricorsivamente.

Lambda Calcolo: Unicità della forma normale e consistenza della beta-equivalenza

$$\forall M, P, Q: M\ \rightarrow \beta\ P\ e\ M\ \rightarrow \beta\ Q, \exists\ R\ :\ P\ \rightarrow \beta\ R\ e\ Q\ \rightarrow \beta\ R$$

Una immediata conseguenza di questo teorema è il seguente corollario: Se un termine ha una forma normale, questa è unica.

Supponiamo, per assurdo, che esista un termine M che abbia due forme normali. Tale cioè che esistano due termini in forma normale N1 e N2 tali che $M \rightarrow \beta\ N1\ e\ M \rightarrow \beta\ N2$. Per il Teorema di Church Rosser deve allora esistere un termine R tale $che\ N1 \rightarrow \beta\ R\ e\ N2 \rightarrow \beta\ R$. Siccome però N1 e N2 sono in forma normale, se $N1 \rightarrow \beta\ R$ deve necessariamente essere che arrivo ad R partendo da N1 con zero passi di riduzione, cioé R è identico a N1. Lo stesso discorso per N2→β R. Questo significa che N1 e N2 sono termini identici, e quindi non è vero che M possa avere due forme normali distinte.

Un altro modo per scrivere il teorema è:

$$M \Rightarrow \beta\ N\ se\ e\ solo\ se\ \exists W\ M\ \Rightarrow \beta\ W\ B \Leftarrow N$$

Se prendiamo ad esempio i lambda termini x e y.y, non è vero che $x \rightarrow \beta\ y.\ y$ è un giudizio valido, altrimenti dovrebbe esistere un termine R tale che $x \rightarrow \beta\ R$ e $y.\ y \rightarrow \beta\ R$, ma questo è impossibile.

Linguaggi Formali

Introduzione

Un linguaggio formale è un insieme, finito o infinito, di *parole* su un determinato *alfabeto*, un insieme finito di simboli.

Per la rappresentazione di un linguaggio formale avente un insieme di parole infinito si utilizzano due metodi:

- *Metodo Generativo*: Si fornisce un insieme finito di strumenti utilizzabili per generazione delle parole del linguaggio;
- *Metodo Riconoscitivo*: Si fornisce un insieme di strumenti utilizzabili per verificare l'appartenenza di una parola a un linguaggio;

Definizione di Alfabeto Σ

Un alfabeto è un *insieme finito non vuoto* di simboli.

$$(sigma)\ \Sigma = \{a, b, c, \dots\}$$

Parola in Σ, Parola Inversa e insieme Σ^*

Dato un al*fabeto* Σ, una *parola su* Σ è definita come una *sequenza finita* di simboli di Σ.

$$\Sigma = \{a, b\} \rightarrow \text{aabbaa} \in \Sigma^*$$

L'insieme di tutte parole appartenenti all'alfabeto Σ viene indicato con $(sigma^{star})$ Σ^*. Si definisce una parola speciale denominata *parola vuota* $(epsilon)$ ϵ che non contiene simboli.

Data una parola x, definiamo la parola inversa di x come $\tilde{x}$:

$$\tilde{x} = \begin{cases} x \; se \; x = \epsilon \; o \; x = a \; [a \in \Sigma] \\ a\tilde{y} \; se \; x = ya \; [y \in \Sigma^*, a \in \Sigma] \\ \forall a \in \Sigma \end{cases}$$

Pur essendo *l'alfabeto un insieme finito di simboli,* è possibile definire un *insieme infinito di parole di lunghezza finita* poiché *la lunghezza delle parole può essere infinita. Per questo* motivo Σ^* è *un insieme infinito.*

Definizione di Linguaggio e di Cardinalità di un linguaggio

Un linguaggio L su un alfabeto Σ è un qualunque sottoinsieme di Σ^*.

- Un linguaggio può essere finito o infinito a seconda del tipo di sottoinsieme che lo rappresenta;
- Dato che $\Sigma^* \subseteq \Sigma^*$, Σ^* stesso è un linguaggio;
- Dato che $\Sigma \subseteq \Sigma^*$, Σ stesso è un linguaggio;
- Dato che $(insieme \; vuoto)$ $\emptyset \subseteq \Sigma^*$, $\emptyset$ è un linguaggio e viene rappresentato con $(lambda)$ Λ;
- Il linguaggio Λ e il linguaggio contenente solo la parola vuota ϵ non sono lo stesso linguaggio:

$$card(\Lambda) = 0 \, , card(\{\epsilon\}) = 1$$

Dato un alfabeto $\Sigma = \{a_1, \ldots, a_n\}$, si definisce *ordinamento lessicografico* delle stringhe di Σ^* *l'ordinamento* < ottenuto stabilendo un qualunque ordinamento tra i caratteri di Σ e definendo l'ordinamento di due stringhe $x, y \in \Sigma^*$ in modo tale che:

$$x < y \leftrightarrow \begin{cases} |x| < |y| \\ |x| = |y| \end{cases}$$

Grazie all'ordinamento lessicografico, possiamo associale alle stringhe di Σ^*una enumerazione e quindi dire che Σ^* è numerabile e ha cardinalità pari all'insieme dei numeri naturali. L'insieme dei linguaggi su Σ non è numerabile.

Definizione di monoide (Monoide sintattico definito su Σ, Operazione di Concatenazione di parole)

Un monoide $(M, \#, e)$ è una *struttura algebrica* in cui è definita *un'operazione interna #*, associativa, rispetto alla quale esiste l'elemento neutro *e*.

- *Operazione interna* significa che l'operazione eseguita su elementi dell'insieme restituirà un elemento sempre appartenente all'insieme:

$$\forall \, a, b \in M \rightarrow a \, \# \, b = c \in M$$

- L'elemento neutro è quell'elemento che:

$$\forall \, x \in M \rightarrow x\#e = e\#x = x$$

Il monoide $(\Sigma^*, \cdot, \epsilon)$ è detto monoide sintattico definito sull'alfabeto Σ poiché è possibile definite l'operazione interna e associativa di *concatenazione di parole* $(\cdot)$ avente come elemento neutro la *parola vuota* ϵ.

Esempio: $\Sigma = \{a, b\} \rightarrow \forall \, x, y, z \in \Sigma^* \rightarrow x = aa, y = bb, z = ab \rightarrow (x \cdot y) \cdot z = x \cdot (y \cdot z) = aabbab \in \Sigma^*$
$L'elemento \; neutro: x \cdot \epsilon = \epsilon \cdot x = x$
$x^n \rightarrow concatenazione \; di \; n \; volte \; di \; x \rightarrow x = ab \rightarrow x^3 = ababab$
$Caso \; speciale \; x^0 \rightarrow x^0 = \epsilon$

L'operazione di concatenazione di parole *non gode della proprietà commutativa:*

$$\Sigma = \{a, b\} \rightarrow \forall \, x, y \rightarrow x \neq y \rightarrow x \cdot y \neq y \cdot x$$

Definizione di lunghezza di una parola basata sul concetto ricorsivo di concatenazione

La lunghezza di una parola $x \in \Sigma^*$ è denotata come $|x|$ ed è definita attraverso un concetto ricorsivo di concatenazione, si definiscono due casi:

$$|\epsilon| = 0 \; (La \; lunghezza \; della \; parola \; vuota \; \text{è} \; pari \; a \; 0)$$

$$|n \cdot a| = |n| + 1 \rightarrow \forall\, n \in \Sigma^*, a \in \Sigma$$

Usando questi due casi è possibile calcolare la dimensione di una parola:

$$Esempio: y = ab \rightarrow |y| = |a| \cdot b = |y| = |a| + 1 = |y| = |\epsilon| \cdot a + 1 = \cdots = 2$$

Definizione di alcune operazioni sui Linguaggi (Unione, Intersezione, Complemento, Concatenazione, Prodotto, Elevazione a Potenza)

Essendo un linguaggio formalmente un sottoinsieme di Σ^*, è possibile applicare le operazioni di insiemistica. Dati due linguaggi $L_1\ e\ L_2$:

- L'unione rappresenta l'insieme di tutte le parole di $L_1\ e\ L_2$:
$$L_1 \cup L_2 = \{n \mid n \in L_1\ o\ n \in L_2\}$$
- L'intersezione rappresenta l'insieme delle parole comuni fra $L_1\ e\ L_2$:
$$L_1 \cap L_2 = \{n \mid n \in L_1\ e\ n \in L_2\}$$
- Il complemento rappresenta l'insieme complementare delle parole del linguaggio L:
$$\bar{L} = \{n \in \Sigma^* \mid n \notin L\}$$
- La concatenazione rappresenta l'insieme delle parole ottenibili dalla concatenazione delle parole di $L_1\ e\ L_2$:
$$L_1 \cdot L_2 = \{n \in\ \Sigma^* \mid \exists\ x_1 \in L_1, \exists\ x_2 \in L_2 \mid n = x_1 \cdot x_2\}$$
  - ❖ Il linguaggio contenente solo la parola vuota è l'elemento neutro nella concatenazione di linguaggi:
  $$L \cdot \{\epsilon\} = \{\epsilon\} \cdot L = L$$
  - ❖ Il linguaggio senza parole rompe il vincolo di esistenza della concatenazione di linguaggi, annullandola:
  $$L \cdot \Lambda = \Lambda$$
- Il prodotto rappresenta l'insieme di coppie ordinate delle parole di $L_1\ e\ L_2$:
$$L_1 \times L_2 = \{(x_1, x_2) \mid x_1 \in\ L_1,\ x_2 \in L_2\}$$
- L'elevazione a potenza di un linguaggio rappresenta una concatenazione ricorsiva:
$$L^h = L \cdot L^{h-1}\ \forall\ h \geq 1$$
  - ❖ Un linguaggio elevato a 0 rappresenta un linguaggio contenente solo la parola vuota:
  $$L^0 = \{\epsilon\} \rightarrow \text{anche nel caso in cui } L = \Lambda$$
  - ❖ In generale, nella concatenazione di linguaggi, $L \nsubseteq L^h\ con\ h \neq 1\ ma\ se\ \epsilon \in L \rightarrow L \subseteq L^h\ \forall\ h \neq\ 0$
  - ❖ L'insieme di tutte le parole di lunghezza *h* di un alfabeto Σ è indicato da: $\Sigma^h = \Sigma \cdot \Sigma \ldots \cdot \Sigma$

Definizione di chiusura di Kleene ($L^*e\ L^+$)

Sia $L \subseteq \Sigma^*$ un linguaggio:

- Si indica *chiusura di Kleene* il linguaggio L^* definito come $\bigcup_{h=0}^{\infty} L^h = L^0 \cup L^1 \ldots \cup L^\infty$
  La *chiusura di Kleene* contiene tutte (infinite) le possibili parole ottenibili concatenando infinite volte le parole di *L.*
- Si indica *chiusura positiva di Kleene* il linguaggio L^+definito come $\bigcup_{h=1}^{\infty} L^h = L^1 \cup L^2 \ldots \cup L^\infty$
  Dato che *chiusura positiva di Kleene* inizia con il *coefficiente h pari a 1*, si ha che:
$$\epsilon \in L^+ \leftrightarrow \epsilon \in L\ \&\ L^* = L^+ \cup \{\epsilon\}$$

Funzionamento di un Riconoscitore (o Automi in generale)

Nel riconoscimento di un linguaggio, l'automa è un dispositivo astratto che data una stringa in input esegue una computazione e, nel caso quest'ultima termini, dirà se la stringa appartiene o meno al linguaggio.

Questo strumento astratto è costituito dalle diverse componenti, alcune essenziali ed altre opzionali:

- Uno o più dispositivi di memoria costituiti da celle denominati *nastri* di lunghezza *infinita* (poiché come abbiamo detto, la lunghezza di una parola può essere infinita) su cui vengono scritte, un simbolo per cella, stringhe di dimensione finita. Le celle non contenenti alcun simbolo contengono un carattere speciale denominato $blank\ \cancel{b}$;
- Una testina per ogni nastro che permette la lettura e/o scrittura di una cella del nastro. Una testina può eseguire spostamenti di una cella per volta a destra o a sinistra;

- Un dispositivo interno di controllo che in ogni istante della computazione assume uno stato di un insieme di stati predefiniti;
- Opzionalmente, dispositivi di memoria ausiliaria su cui immagazzinare dati;

Si può descrivere il funzionamento di un automa come segue:

1. Si scrive la stringa sul nastro, un carattere per cella;
2. Si posiziona la testina sulla prima cella;
3. Si pone il dispositivo di controllo in uno specifico stato denominato *stato iniziale;*

Si inizializza così l'automa in una *configurazione iniziale* da cui potrà partire la computazione che prosegue a passi in cui l'automa può eseguire le seguenti operazioni:

- Scrivere un simbolo sulla cella che la testina sta puntando;
- Spostare la testina di una posizione a destra o sinistra;
- Cambiare stato;

Ad ogni passo della computazione, la configurazione cambia. Una configurazione è quindi una fotografia dell'automa in un determinato momento della computazione, infatti una configurazione descrive:

- Lo stato in cui si trova l'automa;
- Il carattere letto dalla testina;

L'automa viene gestito da un programma interno denominato *funzione di transizione*, che determina i movimenti dell'automa.

Diremo che una stringa appartiene al linguaggio se l'automa termina la sua computazione e l'automa si trova in uno stato particolare denominato *stato finale*.

Introduzione: Automi a stati finiti (ASF)

Gli automi a stati finiti sono particolari riconoscitori in cui:

- La testina può solo leggere e può muoversi solo verso destro;
- Non vi sono memorie ausiliarie;
- La computazione termina sempre, l'automa legge tutta la stringa;

In particolare, gli automi a stati finiti si dividono in due tipologie:

- *Automi a Stati Finiti Deterministici (ASFD):* quando una configurazione ammette al più una configurazione successiva;
- *Automi a Stati Finiti Non Deterministici (ASFND):* quando una configurazione ammette più configurazioni successive;

Automi a Stati Finiti Deterministici (ASFD)

Un *automa a stati finiti deterministico* (ASFD) è definito come una *quintupla* $A = \langle \Sigma, Q, \delta, q_0, F \rangle$ in cui:

- Σ è l'alfabeto di input, un alfabeto finito;
- $Q = \{q_0, q_1, \dots, q_n\}$, è l'insieme finito e non vuoto degli stati che l'automa può assumere durante la computazione;
- $F \subseteq Q$ è l'insieme degli stati finali;
- q_0 è lo stato iniziale;
- δ è la funzione di transizione, che descrive il comportamento dell'automa:

  $$\delta: Qx\Sigma \to Q$$

  La funzione di transizione prende in input una coppia $(stato, simbolo)$ e restituisce uno stato. Tale funzione è una funziona totale poiché è definita per ogni elemento. La funzione di transizione può essere rappresentata come una tabella.

ASFD-ASFND: Rappresentazione tramite Grafo Orientato

Un grafo è un insieme di elementi detti nodi che possono essere collegati fra loro medianti linee denominate archi.

È possibile rappresentare un automa sotto forma di grafo ponendo:

- I nodi del grafo saranno gli stati dell'automa, per convenzione gli stati finali sono doppiamente cerchiati;
- Gli archi uscenti di ogni nodo/stato sono descritti dalla funzione di transizione dell'automa stesso;

ASFD-ASFND: Definizione di configurazione e tipologie

Dato un ASFD $A = \langle \Sigma, Q, \delta, q_0, F \rangle$, una configurazione di A è una coppia $(q, x)\ con\ q \in Q, x \in \Sigma^*$, che può essere:

- *Iniziale* se $q = q_0$;
- *Finale* se $x = \epsilon$;
- *Accettante* se $x = \epsilon, q \in F$;

ASFD-ASFND: Definizione di transizione di configurazione in un passo

Dato un ASFD $A = \langle \Sigma, Q, \delta, q_0, F \rangle$ e due configurazioni $c_1 = (q, x)\ e\ c_2 = (q', y)$ di A, diremo che da c_1 raggiungiamo in un solo passo c_2, denotando $c_1 \vdash c_2$, se e solo se:

1. $x = a \cdot y\ con\ a \in \Sigma\ e\ y \in \Sigma^*$;
2. $\delta(q, a) = q'$;

ASFD-ASFND: Definizione di transizione di configurazione in più passi

Dato un ASFD $A = \langle \Sigma, Q, \delta, q_0, F \rangle$ e due configurazioni $c\ e\ c'$ di A, diremo che da c raggiungiamo in 0 o più passi c', denotando $c \vdash^* c'$, se esistono $c_0, c_1, \dots, c_n\ con\ n \geq 0$ configurazioni tali che:

$$c = c_0, \qquad c' = c_n$$
$$c_i \vdash c_{i+1}\ per\ ogni\ i = 0, \dots, n-1$$

ASFD-ASFND: Terminazione della computazione (computazione di accettazione o di rifiuto)

Se la successione di configurazioni $c_0, c_1, \dots, c_n$ ha lunghezza finita ed è massimale (ovvero non esiste nessuna configurazione *c* tale che $c_n \vdash c$) allora si ha che la computazione termina.

Una computazione che termina può essere:

- *Computazione di accettazione* se termina in una configurazione di accettante;
- *Computazione di rifiuto* se termina in una configurazione finale ma non accettante;

ASFD: Definizione di linguaggio riconosciuto

Il linguaggio riconosciuto da un automa ASFD è così definito:

$$L(A) = \{x \in \Sigma^* \rightarrow (q_o, x) \vdash^* (q, \epsilon)\ con\ q \in F\}$$

- Caso speciale:

$$q_0 \in F \rightarrow (q_0, \epsilon) \vdash^* (q_0, \epsilon)\ quindi\ \epsilon \in L(A)$$

- Per gli automi a stati finiti il concetto di *linguaggio riconosciuto* e *linguaggio accettato* è medesimo poiché l'automa a stati finiti per definizione termina sempre la propria computazione. Per le macchine di Turing, dove non è certa la terminazione della computazione, i concetti non si sovrappongono. Un automa generico accetta un linguaggio quando non è detto che la computazione termina;
- Un linguaggio può essere riconosciuto da diversi automi ma un automa riconosce un solo linguaggio;

ASFD: Definizione di funzione di transizione estesa

La funzione di transizione estesa di un ASFD $A = \langle \Sigma, Q, \delta, q_0, F \rangle$ è la funzione

$$\bar{\delta}: Q x \Sigma^* \rightarrow Q\ definita\ come:$$

- $\bar{\delta}(q, \epsilon) = q$
- $\bar{\delta}(q, x \cdot a) = \delta\big(\bar{\delta}(q, x), a\big)\ con\ x \in \Sigma^*, a \in \Sigma$

Dato uno stato $q \in Q$ e una stringa $x \in \Sigma^*$ in input diremo che $q' = \bar{\delta}(q, x)$ se e solo se la computazione dell'automa a partire dallo stato q e in conseguenza della lettura di x porta l'automa nello stato q'.

ASFD: Definizione di linguaggio riconosciuto (sulla base della funzione di transizione estesa)

Il linguaggio riconosciuto da un automa ASFD $A = \langle \Sigma, Q, \delta, q_0, F \rangle$ è così definito:

$$L(A) = \{x \in \Sigma^* \rightarrow \bar{\delta}(q_0, x) \in F\}$$

ASFD-ASFND: Classe dei linguaggi regolari

Non tutti i linguaggi ammettono un automa a stati finiti che lo riconoscono; infatti, la classe dei linguaggi riconosciuti dagli automi a stati finiti non è infinito.

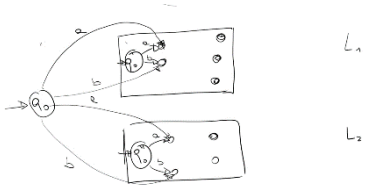
Si definisce la *classe dei linguaggi regolari* R: l'insieme dei linguaggi per cui esiste un ASF che li riconosca.

$$R = \{L \rightarrow L \subseteq \Sigma^* \; ed \; esiste \; un \; automa \; A \; per \; cui \; L = L(A)\}$$

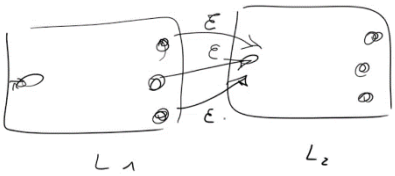
Un linguaggio regolare può avere un automa che lo riconosce, un'espressione regolare che lo denota e una grammatica regolare che lo genera.

Dato un linguaggio regolare esiste solo un unico automa minimale, rispetto al numero di stati, che riconosce il linguaggio.

La classe dei linguaggi regolari è chiusa rispetto alle operazioni di unione, intersezione, complemento, concatenamento e *:



- Se $L1 \; e \; L2$ sono linguaggi regolari, allora $L1 \cap L2$ è regolare: possiamo creare un nuovo automa che incorpora i due automi, si definisce un nuovo stato iniziale che punta alle configurazioni successive agli stati iniziale degli automi;
- Se L è regolare, il suo complemento $\bar{L}$ è regolare: un automa riconosce che il linguaggio L può riconoscere il complemento ponendo $F = Q \backslash F \rightarrow \bar{A}\langle \Sigma, Q, q_0, \delta, Q \backslash F \rangle$. La costruzione funziona solo con ASFD poiché la funziona di transizione è totale, negli ASFND per la stessa stringa si potrebbe avere contemporaneamente una computazione di accettazione una di rifiuto;
- Se $L1 \; e \; L2$ sono linguaggi regolari, allora $L1 \cup L2$ è regolare: per la *legge di De Morgan* $L1 \cap L2 = \overline{(\overline{L1} \cup \overline{L2})}$;
- Se $L1 \; e \; L2$ sono linguaggi regolari, allora $L1 \cdot L2$ è regolare: possiamo creare un nuovo automa che incorpora i due automi, si ha che gli stati finali dell'automa che riconosce A_{L1} puntino allo stato iniziale di A_{L2};
- Se L è regolare, il suo L^* è regolare: gli stati finali dell'automa puntano allo stato iniziale;



Automi a Stati Finiti Non Deterministici (ASFND)

Un *automa a stati finiti non deterministico* (ASFND) è definito come una *quintupla* $A_N = \langle \Sigma, Q, \delta_N, q_0, F \rangle$ in cui:

- Σ è l'alfabeto di input, un alfabeto finito;
- $Q = \{q_0, q_1, \dots, q_n\}$, è l'insieme finito e non vuoto degli stati che l'automa può assumere durante la computazione;
- $F \subseteq Q$ è l'insieme degli stati finali;
- q_0 è lo stato iniziale;
- δ_N è la funzione di transizione, che descrive il comportamento dell'automa:

  $$\delta_N : Qx\Sigma \rightarrow P(Q)$$

  $$P(Q) = \{contiene \; tutti \; i \; possibili \; insiemi \; che \; si \; possono \; costruire \; da \; Q\}$$

  La funzione di transizione prende in input una coppia $(stato, simbolo)$ e restituisce un possibile sottoinsieme di Q. Tale funzione non è una funziona totale per questo motivo se durante la computazione la funzione di transizione restituisce l'insieme vuoto la computazione termina, senza che la stringa in input sia stata letta completamente.

La computazione di un ASFD ha un numero di passi pari alla lunghezza della stringa, mentre nei ASFND la computazione ha un numero di passi al più pari alla lunghezza della stringa.

Dato che una configurazione ammette più configurazioni successive, quando l'automa incontra più possibili configurazioni successive la computazione si biforca prendendo strade differenti.

Nella rappresentazione a grafo, un ASFND presenta nodi con più archi uscenti.

ASFND: Definizione di linguaggio riconosciuto

Il linguaggio riconosciuto da un automa ASFND è così definito:

$$L(A) = \{x \in \Sigma^* \rightarrow (\{q_0\}, x) | -^* (p, \epsilon) \; con \; p \subseteq Q \; e \; p \cap F \neq \emptyset\}$$

ASFND: Definizione di funzione di transizione estesa

La funzione di transizione estesa di un ASFND $A_N = \langle \Sigma, Q, \delta_N, q_0, F \rangle$ è la funzione

$$\bar{\delta}: Qx\Sigma^* \rightarrow P(Q) \; definita \; come:$$

- $\bar{\delta}(q, \epsilon) = \{q\} \; \forall \; q \in Q$
- $\bar{\delta}(q, x \cdot a) = \bigcup_{p \in \bar{\delta}(q,x)} \delta(p, a) \;\; \forall \; x \in \Sigma^*, a \in \Sigma$

ASFND: Definizione di linguaggio riconosciuto (sulla base della funzione di transizione estesa)

Il linguaggio riconosciuto da un automa ASFND $A = \langle \Sigma, Q, \delta_N, q_0, F \rangle$ è così definito:

$$L(A) = \{x \in \Sigma^* \rightarrow \; \bar{\delta}(q_0, x) \cap F \neq \emptyset\}$$

ASFD-ASFND: Relazione fra ASFD e ASFND nel riconoscimento di un linguaggio

Dato un ASFD che riconosce un linguaggio L, esiste corrispondentemente un ASFND che riconosce lo stesso linguaggio L; viceversa, dato un ASFND che riconosce un linguaggio L', esiste un ASFD che riconosce lo stesso linguaggio L'.

Definizione e Dimostrazione del Pumping Lemma

Il Pumping Lemma ci permette di dire che un linguaggio non è un linguaggio regolare, ovvero riconosciuto da un automa, senza dover ricorrere agli automi.

Il Pumping Lemma è una proprietà necessaria ma non sufficiente per i linguaggi regolari:

- Se un linguaggio è regolare, allora il linguaggio soddisfa il pumping lemma;
- Se un linguaggio soddisfa il pumping lemma, non è detto che il linguaggio è regolare;

Definizione: Per ogni $linguaggio \; regolare \; L$, esiste una $costante \; n$, tale che $z \in L \; e \; |z| \geq n$ allora possiamo scrivere che: $z = uvw \in L, |uv| \leq n \; e \; |v| \geq 1 \; con \; v \neq \epsilon$ e ottenere che $uv^i w \in L \; \forall i \geq 0$.

Dimostrazione:

Se L è $un \; linguaggio \; regolare$, allora esiste $A = \langle \Sigma, Q, \delta, q_0, F \rangle$ che riconosce tale linguaggio, ovvero $L = L(A)$.

Supponiamo che $|Q| = n$ e $z \in L \; con \; |z| \geq n$, durante la computazione si avrà la sequenza di stati $q_{i0}, q_{i1}, \dots, q_{ik} \; con \; q_{i0} = q_0 \; e \; q_{ik} \in F$, dato che è passati per $k + 1$ stati e che $k + 1 > k \geq n$ si ha la certezza che ci sono due stati che coincidono.

Questo significa che esistono due indici $r \; ed \; s$ con $0 \leq r < s \leq n$ tali che $\bar{\delta}(q_0, z_r) = q_{ir} = q_{is} = \bar{\delta}(q_0, z_s)$, con $z_h \; si \; indica \; il \; prefisso \; di \; z \; di \; linghezza \; h.$

Poniamo: $u = z_r, \;\; uv = z_s \; e \; uvw = z$. Si ha che: $|uv| = |z_s| = s \leq n, |u| = |z_r| = r < s$

Dato che: $|uv| = |u| + |v| \rightarrow |v| \geq 1.$

Passo Base: $per \; i = 0 \; bisogna \; dimostrare \; che \; uv^0 w = u\epsilon w = uw \in L$:

$$\bar{\delta}(q_0, uw) = \; \bar{\delta}\big(\bar{\delta}(q_0, u), w\big) = \; \bar{\delta}\big(\bar{\delta}(q_0, z_r), w\big) = \bar{\delta}\big(\bar{\delta}(q_0, z_s), w\big) = \bar{\delta}\big(\bar{\delta}(q_0, uv), w\big) = \bar{\delta}(q_0, uvw) = \bar{\delta}(q_0, z) \in F$$

Passo Induttivo: $per \; i > 0 \; supponiamo \; che \; la \; proprietà \; per \; i - 1 \; sia \; vero, ovvero \; uv^{i-1} w \in L$:

$$\bar{\delta}(q_0, uvw) = \bar{\delta}(q_0, uvv^{i-1}w) = \bar{\delta}(\bar{\delta}(q_0, uv), v^{i-1}w) = \bar{\delta}(\bar{\delta}(q_0, z_s), v^{i-1}w) = \bar{\delta}(\bar{\delta}(q_0, z_r), v^{i-1}w)$$
$$= \bar{\delta}(\bar{\delta}(q_0, u), v^{i-1}w) = \bar{\delta}(q_0, uv^{i-1}w) \in F$$

- Soddisfare il pumping lemma significa che $\exists n\ \forall z \in L\ con\ |z| \geq n\ tale\ che\ \exists\ u, v, w\ con\ |uv| \leq n, v \neq \epsilon\ e\ z = uvw\ si\ ha\ che\ \forall i \geq 0\ tale\ che\ uv^i w \in L$;
- Non soddisfare il pumping lemma significa che $\forall n\ \exists\ z \in L\ con\ |z| \geq n\ tale\ che\ \forall\ u, v, w\ con\ |uv| \leq n, v \neq \epsilon\ e\ z = uvw\ si\ ha\ che\ \exists i \geq 0\ tale\ che\ uv^i w \notin L$;

Definizione di Espressione Regolare

Dato un alfabeto Σ e dato l'insieme di simboli $\{+, \cdot, (,), *, \emptyset\}$, si definisce *espressione regolare su* Σ una stringa $r \in (\Sigma \cup \{+, \cdot, (,), *, \emptyset\})^*$ tale che valgano le seguenti condizioni:

1. $r = \emptyset$
2. $r = a\ con\ a\ \in \Sigma$
3. $r = (s + t)\ oppure\ r = (s \cdot t)\ oppure\ r = s^*\ con\ s\ e\ t\ espressioni\ regolari$

Nelle espressioni regolari è possibile omettere le parentesi se si utilizza la priorità:

$$* \rightarrow \cdot \rightarrow +$$

Definizione di linguaggio rappresentato da un'espressione regolare

Data un'espressione regolare r, il linguaggio denotato da r $L(r)$ è così definito:

1. Se $r = \emptyset$ allora $L(r) = \Lambda$
2. Se $r = a \in \Sigma$ allora $L(r) = \{\mathrm{a}\}$
3. Se $r = (s + t)$ allora $L(r) = \mathrm{L(s)} \cup \mathrm{L(t)}$
4. Se $r = (s \cdot t)$ allora $L(r) = \mathrm{L(s)} \cdot \mathrm{L(t)}$
5. Se $r = s^*$ allora $L(r) = \mathrm{L(s)}^*$

Espressioni regolari diverse possono rappresentare lo stesso linguaggio.

Relazione fra automi ed espressioni regolari

Tutto ciò che posso riconoscere con un automa è rappresentabile attraverso l'uso di espressioni regolari.

Per questo motivo possiamo dire che *la classe dei linguaggi regolari coincide con la classe dei linguaggi rappresentati da espressioni regolari.*

- Se L è $regolare \rightarrow \exists\ r\ tale\ che\ L = L(r)$
- Se $L = L(r)$ (rappresentato da una espressione regolare)$\rightarrow \exists\ ASFD\ tale\ che\ L = L(A)$

Inoltre:

- Dato un automa A è possibile definire un'espressione regolare utilizzando il *teorema di Kleene;*
- Data un'espressione regolare *r*, è possibile costruire un automa attraverso la *costruzione di Thompson;*

Confronto di espressioni regolari

Dato che espressioni regolari diverse possono rappresentare lo stesso linguaggio e che non esiste *una forma normale* con cui rappresentarle, per confrontare due espressioni regolari, ovvero sapere se rappresentano lo stesso linguaggio, è necessario confrontare gli *automi (minimali)* ottenuti attraverso la *costruzione di Thompson.*

Definizione di Grammatica

Una grammatica è uno strumento generativo, ovvero genera linguaggi.

Una $grammarica\ G$ è una qualunque quadrupla $G = (V_T, V_N, P, S)$ in cui:

- V_N è un insieme infinito e non vuoto detto *alfabeto terminale*;
- V_T è un insieme infinito e non vuoto detto *alfabeto non terminale*;
- P è l'insieme finito e non vuoto delle regole di produzione, ovvero $coppie\ (a, \beta)$ indicate con $a \rightarrow \beta$ in cui:
$$a \in (V_T \cup V_N)^* \cdot V_N \cdot (V_T \cup V_N)^*\ (una\ stringa\ contiene\ almeno\ 1\ simbolo\ non\ terminale)$$

$$\beta \in (V_T \cup V_N)^* \ (una\ stringa, anche\ vuota, contenente\ simboli\ terminale\ e\ non)$$

- ❖ Possiamo affermare che: $P \subseteq ((V_T \cup V_N)^* \cdot V_N \cdot (V_T \cup V_N)^*) \times (V_T \cup V_N)^*$
- ❖ Se $P = \emptyset$ allora la grammatica genere il linguaggio vuoto Λ;
- ❖ La regola di produzione $\alpha \to \epsilon$ prende il nome di $\epsilon_{produzione}$;

- $S \in V_N$ è il simbolo iniziale;

Definizione di Derivazione Diretta

Sia $G = (V_T, V_N, P, S)$ una grammatica, se $a\beta\gamma \in (V_T \cup V_N)^*$ *e* $\beta \to S \in P$ diremo che $a\beta\gamma$ deriva direttamente $aS\gamma$ con la notazione:

$$a\beta\gamma \underset{G}{\Rightarrow} aS\gamma$$

Definizione di Derivazione Non Banale

Sia $G = (V_T, V_N, P, S)$ una grammatica, se $\alpha, \beta \in (V_T \cup V_N)^*$ diremo che α deriva in modo non banale β con la notazione:

$$\alpha \overset{+}{\underset{G}{\Rightarrow}} \beta$$

Se $\exists\ a_0, a_1, \dots, a_n\ con\ n \geq 1$ tali che:

$$\alpha = a_0 \underset{G}{\Rightarrow} a_1 \underset{G}{\Rightarrow} \dots \underset{G}{\Rightarrow} a_n = \beta$$

- La derivazione non banale può essere vista come la chiusura transitiva della derivazione diretta;
- La derivazione diretta può essere vista come un caso particolare di derivazione non banale in cui $n = 1$;

Definizione di Derivazione (generica)

Sia $G = (V_T, V_N, P, S)$ una grammatica, se $\alpha, \beta \in (V_T \cup V_N)^*$ diremo che α deriva β con la notazione:

$$\alpha \overset{*}{\underset{G}{\Rightarrow}} \beta$$

Se $\exists\ a_0, a_1, \dots, a_n\ con\ n \geq 0$ tali che:

$$\alpha = a_0 \underset{G}{\Rightarrow} a_1 \underset{G}{\Rightarrow} \dots \underset{G}{\Rightarrow} a_n = \beta$$

- La derivazione generica può essere vista come la chiusura transitiva e riflessiva della derivazione diretta;

Definizione di Derivazione in K passi

Sia $G = (V_T, V_N, P, S)$ una grammatica, se $\alpha, \beta \in (V_T \cup V_N)^*$ diremo che α deriva β in K passi con la notazione:

$$\alpha \overset{K}{\underset{G}{\Rightarrow}} \beta$$

Se $\exists\ a_0, a_1, \dots, a_K$ tali che:

$$\alpha = a_0 \underset{G}{\Rightarrow} a_1 \underset{G}{\Rightarrow} \dots \underset{G}{\Rightarrow} a_K = \beta$$

Definizione di Forma di Frase

Sia $G = (V_T, V_N, P, S)$ una grammatica, si definisce *forma di frase* una qualunque stringa $\alpha \in (V_T \cup V_N)^*$ tale che:

$$S \overset{*}{\underset{G}{\Rightarrow}} \alpha$$

Definizione di Linguaggio generato da una grammatica

Sia $G = (V_T, V_N, P, S)$ una grammatica, il linguaggio generato dalla grammatica è:

$$L(G) = \{x \in V_T^* \to S \overset{*}{\underset{G}{\Rightarrow}} x\}$$

Diverse grammatiche possono generare lo stesso linguaggio. Due grammatiche si definiscono equivalenti se:

$$L(G_1) = L(G_2)$$

$$x \in L(G_1) \leftrightarrow x \in L(G_2)\ allora\ L(G_1) \subseteq L(G_2)\ e\ viceversa$$

Grammatiche Regolari (Tipo 3)

Le grammatiche regolari sono quelle grammatiche che permettono solo regole di produzione del tipo:

$$A \to \beta\ con\ A \in V_N\ e\ \beta \in \mathrm{V_T} \cup (V_T \cdot V_N)$$

Ovvero a sinistra si può avere solo un simbolo non terminale mentre a destra un simbolo terminale o una concatenazione di un simbolo terminale e uno non terminale. Non si possono avere $\epsilon\_produzioni$.

La classe dei linguaggi generata dalle grammatiche regolari coincide con la classe dei linguaggi regolari, perciò si affermare che:

- Data una grammatica regolare, esiste un ASFND che riconosce il linguaggio che essa genera;
- Dato un ASFND esiste una grammatica regolare che genera il linguaggio che esso riconosce;

Grammatiche Context-Free (Tipo 2)

Le grammatiche context-free sono quelle grammatiche che permettono solo regole di produzione del tipo:

$$A \to \beta\ con\ A \in V_N\ e\ \beta \in (V_T \cup V_N)^+$$

Ovvero a sinistra si può avere solo un simbolo non terminale mentre a destra stringa contenente caratteri terminali e non terminali. Non si possono avere $\epsilon\_produzioni$.

Le grammatiche context-free sono più potenti delle grammatiche regolari poiché la classe dei linguaggi è maggiore.

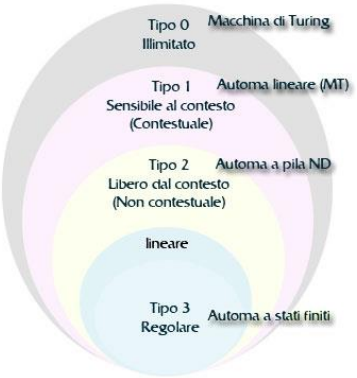
I linguaggi generati da grammatiche di tipo 2 sono riconosciuti da *automi a pila*.

Grammatiche Context-Sensitive (Tipo 1)

Le grammatiche context-sensitive sono quelle grammatiche che permettono solo regole di produzione del tipo:

$$a \to \beta\ con\ a \in (V_T \cup V_N)^* \cdot V_N \cdot (V_T \cup V_N)^*, \beta \in (V_T \cup V_N)^+\ e\ |a| \leq |\beta|$$

Ovvero a sinistra si ha una stringa che contiene almeno un carattere non terminale mentre a destra una stringa contenente simbolo terminali e non terminali con il vincolo che la lunghezza della stringa di sinistra sia inferiore o uguale a quella di destra. Non si possono avere $\epsilon\_produzioni$.



A differenza delle grammatiche di tipo 2, dove la derivazione viene sempre eseguita, le grammatiche context-sensitive permettono la derivazione se e solo se viene soddisfatto un pattern.

Le grammatiche di tipo 1 sono più potenti delle grammatiche tipo 2 poiché la classe dei linguaggi è maggiore.

I linguaggi generati da grammatiche di tipo 1 sono riconosciuti da *automi lineari*.

Grammatiche Senza Restrizioni (Tipo 0)

Le grammatiche senza restrizioni sono quelle grammatiche che permettono regole di produzione del tipo:

$$a \to \beta\ con\ a \in (V_T \cup V_N)^* \cdot V_N \cdot (V_T \cup V_N)^*, \beta \in (V_T \cup V_N)^*$$

Ovvero a sinistra si ha una stringa che contiene almeno un carattere non terminale mentre a destra una stringa contenente simbolo terminali e non terminali oppure la stringa vuota, permettendo così di avere $\epsilon\_produzioni$.

Le grammatiche senza restrizioni sono più potenti delle grammatiche context-sensitive poiché la classe dei linguaggi è maggiore.

I linguaggi generati da grammatiche di tipo 0sono riconosciuti da macchine di Turing.

Definizione di Linguaggio strettamente di tipo N

Un $linguaggio\ L$ è strettamente di tipo n se esiste una *grammatica di tipo* n che lo genera e non esiste una grammatica di tipo $m, con\ m > n$, che lo genera.

Grammatica: Introduzione alle Forme Normali

Il confronto di due grammatiche può essere un problema ad elevata complessità. Portare le grammatiche in una forma normale diminuisce la complessità.

Grammatica: Forma Normale di Chomsky

La *forma normale di Chomsky* prevede che la grammatica abbia solo regole di produzione del tipo:

$$A \to BC, A \to a \; con \; A, B, C \in V_N \; e \; a \in V_T$$

Ogni grammatica context-free può essere trasformata in una grammatica in forma di Chomsky equivalente.

Grammatica: Forma Normale di Greibach

La *forma normale di* Greibach prevede che la grammatica abbia solo regole di produzione del tipo:

$$A \to ax \; con \; A \in V_N \; , a \in V_T \;\; e \; x \in V_N^*$$

Dato che la derivazione ha come primo simbolo sempre un simbolo terminale, le grammatiche in forma normale di Greibach non hanno ricorsione sinistra.

Grammatica: Forma Normale di Backus (BNF)

La forma normale di Backus prevede le seguenti rappresentazioni:

1. I simboli non terminali vengono rappresentati con $< espressione >, < identificatore > \in V_N$;
2. La freccia di derivazione viene sostituita: $\alpha \to \beta \; diventa \; \alpha ::= \beta$;
3. $A ::= x|xy|xyy|xyyy$ può essere ristretto in $A ::= x\{y\}^3$
4. $A ::= xy|y$ può essere ristretto in $A ::= [x]y$

Grammatica: Albero di Derivazione

Possiamo rappresentare una derivazione di una grammatica sotto forma di albero. La stringa generata dalla derivazione può essere ottenuta leggendo le foglie dell'albero da sinistra a destra.

Un albero di derivazione può essere associato a più derivazione poiché è possibile applicare regole di derivazione in tempi diversi sullo stesso elemento.

Per questo motiva di adotta la conversione della *derivazione sinistra,* ovvero durante una derivazione si applica *la regola di derivazione del simbolo non terminale più a sinistra.*

Un albero di derivazione può essere associato a più derivazioni poiché posso applicare in tempi diversi le regole di attivazione allo stesso elemento.

Definizione di Grammatica e Linguaggio ambigui

Una $grammatica \; G$ è detta ambigua se esiste una stringa $x \in L(G)$ che ammette due diverse *derivazioni sinistre.*

Un $linguaggio \; L$ si dice *inerentemente ambiguo* se ogni grammatica che lo genera è ambigua.

Macchine di Turing (MT)

Le macchine di Turing permettono di riconoscere linguaggi oppure di calcolare delle funzioni. Possiamo dire che qualunque programma che esegue un calcolo può essere rappresentato con una macchina di Turing.

Le macchine di Turing sono particolari riconoscitori in cui:

- La testina può leggere e scrivere e muoversi a destra e sinistra;
- All'interno del nastro è possibile trovare il carattere di blank ~~*b;*~~
- La computazione potrebbe non terminare mai;

Una macchina di Turing deterministica MTD è una sestupla $m = (\Gamma, \beta, \mathrm{Q}, \mathrm{q}_0, F, \delta)$ in cui:

- Γ è l'alfabeto dei simboli del nastro;
- $\beta \notin \Gamma$ è il simbolo speciale detto *blank*;
- *Q è l'insieme finito degli stati;*

- $q_0 \in Q$ *è lo stato iniziale;*
- $F \subseteq Q$ *è l'insieme degli stati finali;*
- δ *è la funzione di transizione definita come:*

$$\delta: (Q \backslash F) \times (\Gamma \cup \{\beta\}) \rightarrow Q \times (F \cup \{\beta\} \times \{d, s, i\})$$

La funzione di transizione prende in input una coppia $(stato\ non\ finale, simbolo)$ e restituisce una t-upla $(stato, carattere\ scritto\ nella\ cella, movimento\ testina)$.

Quando la macchina arriva in uno stato finale, la computazione si ferma. Se la funzione di transizione non è definita, la computazione si ferma. Altrimenti la computazione continua all'infinito.

MT: Definizione di configurazione e tipologie

Dato una MTD m= $(\Gamma, \beta, Q, q_0, F, \delta)$, una configurazione di *m* è una stringa xqy con:

- $x \in \Gamma \cdot \bar{\Gamma}^* + \{\epsilon\}\ dove\ \bar{\Gamma} = \Gamma \cup \{\beta\}$;
- $q \in Q$;
- $y \in \bar{\Gamma}^* \cdot \Gamma + \{\beta\}\ dove\ \bar{\Gamma} = F \cup \{\beta\}$;

Si definiscono quindi due tipologie di configurazione:

- *Iniziale* se $x = \epsilon,\ q = q_0\ e\ y \in \Gamma^+ \cup \{\beta\}$;
- *Finale* se $q \in F$;

MT: Definizione di stringa accettata

Dato una MTD m= $(\Gamma, \beta, Q, q_0, F, \delta)$ e un alfabeta $\Sigma \subseteq \Gamma$, si dice che $x \in \Sigma^*$ è accettata dalla macchina di Turing se e solo se la computazione termina in uno stato finale.

MT: Definizione di linguaggio riconosciuto

Dato una MTD m= $(\Gamma, \beta, Q, q_0, F, \delta)$, diremo che riconosce $L \subseteq \Sigma^*$ se:

- $\forall x \in \Sigma^*$ la computazione termina;
- $Se\ x \in L$ la computazione termina in uno stato finale;
- $Se\ x \notin L$ la computazione termina in uno stato non finale;

MT: Definizione di linguaggio accettato

Dato una MTD m= $(\Gamma, \beta, Q, q_0, F, \delta)$, diremo che accetta $L \subseteq \Sigma^*$ se:

- $\forall\ x \in L$ la computazione termina in uno stato finale;
- $\forall\ x \notin L$ la computazione non termina o termina in uno stato non finale;

MT: Definizione di linguaggio decidibile e semi-decidibile

Un linguaggio è detto decidibile secondo Turing, T-decidibile, se esiste una macchina di Turing che lo riconosce.

Un linguaggio è detto semi-decidibile secondo Turing, T-semidecidibile, se esiste una macchina di Turing che lo accetta.

Codici e Rappresentazione delle informazioni

Stringhe vs Numeri

Una delle differenze tra il modello computazionale delle Macchine di Turing e quello delle URM è che nel primo modello vengono manipolate stringhe di simboli mentre nel secondo caso vengono manipolati numeri naturali. Quindi, mentre domini e codomini delle funzioni calcolate a URM sono sempre i Numeri Naturali, le macchine di Turing possono calcolare funzioni che hanno come dominio (e codominio) qualsiasi insieme i cui elementi si possano rappresentare come stringhe di caratteri. I domini (e codomini) delle funzioni calcolabili da macchine di Turing sono quindi tutti quegli insiemi per i quali sia possibile definire una funzione di codifica e decodifica.

Poiché i numeri naturali (che indichiamo con N) si possono codificare con stringhe sull'alfabeto {0,1}, le macchine di Turing possono lavorare anche sui numeri naturali, oltre che su molti altri domini e codomini.

Codifica dell'informazione

Codificare un'informazione significa rappresentare un'informazione all'interno di un sistema di calcolo mediante l'utilizzo di codici comprendibili e manipolabili dal calcolatore.

Nel processo di codifica, individuiamo:

- L'insieme finito o infinito *X*, che rappresenta gli oggetti o le informazioni che vogliamo trattare;
- L'insieme finito *A,* un qualsiasi alfabeto finito di simboli, e la sua *chiusura di Kleene* A^**;*

Definiamo una funzione iniettiva di codifica: $cod: X \to A^*$

ed una relativa funzione di decodifica: $decod: A^* \to X \cup \{errore\}$

Tali che: $\forall y \in X \to decod\big(cod(y)\big) = y$

L'insieme $\{X, A, cod, decod\}$ formano un Codice per la rappresentazione di elementi di X.

Codifica di un numero in complemento a una base in P posizioni

Per codificare un numero in complemento alla base *B in P posizioni si applica:*

$$cod^p_{ZC\ B}(r)$$

- $r > 0 = cod^p_{N\ B}(r)$
- $r < 0 = cod^p_{N\ B}(B^P - |r|)$

Il numero rappresentato è negativo se e solo se il bit più a sinistra è 1.

Il range di numero rappresentabili è il seguente: $[-B^{P-1}, B^{P-1} - 1]$.

Macchine Astratte

Definizione di macchina astratta e del suo linguaggio

Una macchina astratta (Imperativa) è un insieme di strutture dati e algoritmi in grado di memorizzare ed eseguire programmi. Le componenti principali di una macchina astratta sono:

- Una memoria, destinata a contenere il programma che deve essere eseguito e i dati su cui si sta operando;
- Un insieme di operazioni primitive utili all'elaborazione dei dati primitivi;
- Un insieme di operazioni e strutture dati che gestiscono il flusso di controllo, ovvero che governano l'ordine secondo il quale le operazioni e le istruzioni, descritte dal programma, vengono eseguite;
- Un insieme di operazioni e strutture dati per il controllo del trasferimento dei dati, che si occupa di recuperare gli operandi e memorizzare i risultai delle varie istruzioni;
- Un insieme di operazioni e strutture dati per la gestione della memoria;
- Un interprete, fornisce la capacità di eseguire i programmi coordinando il lavoro delle altre parti della macchina astratta eseguendo un semplice ciclo FETCH/EXECUTE, finché non viene eseguita una particolare istruzione primitiva (detta HALT) che ne provoca l'arresto immediato. Il processo eseguito dall'interprete è suddiviso in fasi:
  - ❖ Inizialmente avviene l'acquisizione, mediante l'uso delle strutture dati preposte al controllo della sequenza, della prossima istruzione da eseguire (FETCH);
  - ❖ L'istruzione viene decodificata e si acquisiscono i suoi eventuali operandi, ricorrendo al controllo sul trasferimento dei dati;
  - ❖ Viene eseguita l'opportuna operazione primitiva e l'eventuale risultato viene memorizzato;
  - ❖ Se l'operazione eseguita non è quella che fa arrestare l'interprete, il processo ricomincia;

Il linguaggio macchina (LM) di una macchina astratta M è il linguaggio in cui si esprimono tutti i programmi interpretabili dall'interprete di M. Spesso però è più conveniente pensare ad un programma in termini di stringhe di caratteri. L'insieme di queste versioni dei programmi forma il cosiddetto LMEST (Linguaggio Macchina ESTerno).

Il programma espresso in forma direttamente memorizzabili nella macchina e la sua versione mnemonica sono due rappresentazioni dello stesso oggetto: il programma astratto. Il compito di realizzare la conversione dal linguaggio

LMEST al linguaggio LM è svolto dal LOADER (così detto perché carica il programma nella memoria della macchina astratta).

Così come data una macchina astratta è definito un linguaggio di programmazione (il suo linguaggio macchina), analogamente dato un linguaggio di programmazione è definita una macchina astratta che ha il dato linguaggio come suo linguaggio macchina.

È da notare che gli High Level Language o HLL definiscono macchine astratte tanto più complesse quanto maggiore è la potenza espressiva del linguaggio in questione. Di conseguenza la realizzazione in hardware è opportuna solo per macchina astratte relativamente semplici, mentre nel caso di una macchina astratta associata ad un linguaggio di alto livello questa scelta è poco conveniente, e risulta preferibile una realizzazione non hardware.

Realizzazione delle Macchine Astratte

Affinché i programmi scritti in un dato linguaggio di programmazione L possano essere eseguiti, è necessario che la corrispondente macchina astratta venga implementata. Vi sono 3 tecniche:

- realizzazione hardware: si realizzano in hardware tutte le componenti della macchina. In linea di principio tale tecnica è sempre possibile ma è prevalentemente usata solo per macchina astratte relativamente semplici a causa di considerazioni di carattere pratico, quali la scarsa flessibilità della realizzazione in hardware e l'elevato costo di progettazione e realizzazione in hardware di componenti dalle funzionalità molto complesse;
- interpretazione (emulazione): basandosi su una macchina ospite già realizzata, si realizzano tutte le componenti della macchina (gli algoritmi e le strutture dati) che si vuole realizzare mediante programmi e strutture dati della macchina ospite; in due modalità:
    - ❖ emulazione via firmware: la macchina ospite è realizzata in hardware e i suoi programmi risiedono in una memoria di sola lettura ad alta velocità. Tale macchina ospite è detta micro-programmata e i suoi programmi vengono detti microprogrammi. Questa modalità permette alla macchina realizzata di avere prestazioni al pari della realizzazione hardware a discapito della flessibilità in caso di modifiche o estensioni;
    - ❖ emulazione via software, non vi è nessun vincolo su come la macchina ospite sia realizzata, potrebbe quindi trattarsi di un'altra macchina anch'essa emulata e così via;
- traduzione (compilazione): si basa sulla idea di tradurre l'intero programma scritto nel linguaggio della macchina che si vuole realizzare in un programma funzionalmente equivalente scritto nel linguaggio della macchina ospite. Il compito di eseguire tale traduzione è eseguito da un programma detto compilatore.

La differenza di potenza espressiva fra una macchina astratta che vogliamo realizzare e la macchina ospite è detta semantic gap, spesso talmente grande che è opportuno implementare una o più macchine intermedie.

Nella realizzazione di una macchina astratta su una macchina ospite, tipicamente, non si effettua mai una pura traduzione o una pura interpretazione (emulazione software) poiché:

- La pura interpretazione potrebbe risultare insoddisfacente a causa della scarsa efficienza della macchina realizzata, causata dall'emulazione software delle sue componenti;
- La pura traduzione, in caso di semantic gap elevato, porterebbe alla creazione di programmi per la macchina ospite che risulterebbero troppo grossi e magari lenti;

Una soluzione a questa problematica è l'introduzione di una macchina astratta intermedia che esegue la compilazione nel proprio linguaggio dei programmi della macchina astratta che si vuole realizzare, che a sua volta vengono eseguiti per interpretazione sulla macchina ospite.

Nel progetto della macchina intermedia bisogna tenere conto di due requisiti di efficienza:

- velocità di simulazione della macchina intermedia sulla macchina ospite;
- compattezza dei programmi compilati della macchina intermedia;

Per soddisfare tali requisiti, tipicamente la macchina intermedia è un'estensione della macchina ospite, nel senso che ne condivide l'interprete, e ne potenzia alcuni aspetti.

Organizzazione a livello di un sistema di calcolo

Un tipico computer moderno può essere pensato come una serie di macchine astratte realizzate una sopra l'altra, ciascuna in grado di fornire funzionalità via via più potenti. Possiamo immaginare una decomposizione in livelli:

- Livello -2: il livello della fisica dello stato solido;
- Livello -1: il livello dell'elettronica circuitale;
- Livello 0: è costituito da porte logiche, i cui raggruppamenti formano i registri e le memorie;
- Livello 1: è costituito da gruppi di registri e circuiti digitali che realizzano funzionalità aritmetico-logiche;
- Livello 2: è il livello a cui le istruzioni macchina operano;
- Livello 3: è il livello del Sistema Operativo, fra i tanti livelli di astrazione offerti vi sono il processo, il filesystem e la memoria virtuale;
- Livello 4: è il livello dell'assembly;
- Livello 5: il livello in cui troviamo gli HLL: C, Java, C++, Haskell, ect…

Semantica Formale

Fornire la definizione matematica formale della semantica di un linguaggio di programmazione ha i seguenti vantaggi:

- Minore ambiguità: rende il comportamento del linguaggio più comprensibile;
- Sinteticità: notazioni e concetti matematici posso descrivere un linguaggio in maniera più comprensibile e sintetica;
- Argomentazioni formali: una semantica formale ci permette di specificare e mettere alla prova le proprietà di un programma;

Uno svantaggio della semantica formale è che spesso, cercando di descrivere le funzionalità di un linguaggio moderno, porta alla creazione di modelli matematici così complessi da rendere la semantica formale difficile da comprendere. Per questo motivo pochi linguaggi ne fanno uso.

Per formalizzare la semantica di un linguaggio vi sono 3 approcci:

- Semantica Operazionale Strutturata: descrive come un programma verrebbe eseguito in una macchina astratta. Consiste in regole come $\frac{A_1 \dots A_n}{A}$ che significa: A è $vera\ se\ tutte\ le\ A_1 \dots A_n\ premesse\ sono\ vere$: