Fondamenti dell'Informatica
(lucidi a.a. 2017-2018)

Agostino Dovier, Roberto Giacobazzi, Roberto BAGNARA
Università di Udine
Università di Verona
Università di Parma

TEOREMA DI CANTOR

Teorema (Cantor)**.** *Sia S un insieme.* $|S| < |\wp(S)|$.

Per assurdo assumiamo esista una funzione $f\colon S \to \wp(S)$ biiettiva. Sia $A = \left\{ x \in S \;\middle|\; x \notin f(x) \right\}$. Poiché $A \in \wp(S)$ e f è per ipotesi suriettiva, deve esistere $a \in S$ tale che $f(a) = A$. Ora, chiediamoci se a appartenga o meno ad A:

- se $a \in A$ allora, per definizione di A, $a \notin f(a) = A$;
- se $a \notin A = f(a)$ allora, per definizione di A, $a \in A$.

In entrambi i casi si giunge ad una contraddizione: l'unica ipotesi che abbiamo fatto è quella dell'esistenza di una f biiettiva, che pertanto non può essere vera.

SIMBOLI

- Un simbolo è un'entità primitiva astratta non meglio definita.
- Per ciò che ci concerne, un simbolo è atomico: se anche avesse una struttura interna, noi non la osserviamo.
- Le uniche caratteristiche rilevanti dei simboli:
  - l'identità, in modo da poter distringuere un simbolo dall'altro;
  - la possibilità, almeno concettuale, di essere giustapposti (posti l'uno accanto all'altro in modo da formare una sequenza [unidimensionale]).
- Esempi: lettere e caratteri numerici; ideogrammi; punti, linee e spazi nell'alfabeto Morse; bandiere per segnalazioni nautiche.

STRINGHE

- Una stringa (o parola) è una sequenza finita di simboli giustapposti
- Ad esempio, se a, b e c sono simboli, $abcba$ è una stringa.
- La lunghezza di una stringa w è il numero di occorrenze di simboli in w e si denota con $|w|$.
- Ad esempio, $|abcba| = 5$.
- La stringa vuota ε è costituita da zero simboli: $|\varepsilon| = 0$.
- Sia $w = a_1 \cdots a_n$ una stringa (notare che, se $n = 0$, allora $w = \varepsilon$). Ogni stringa della forma:
  - $a_1 \cdots a_j$, con $j \in \{0, 1, \ldots, n\}$ è detta prefisso di w;
  - $a_i \cdots a_n$, con $i \in \{1, \ldots, n, n+1\}$ è detta suffisso di w;
  - $a_i \cdots a_j$, con $i, j \in \{1, \ldots, n\}$, detta sottostringa di w.
- Si noti che, ovviamente, abbiamo $a_i \cdots a_j = \varepsilon$ se $i > j$.
- Si noti inoltre che ε è sia prefisso che suffisso che sottostringa di w.

STRINGHE

- I prefissi di abc sono ε, a, ab, e abc.
- I suffissi di abc sono ε, c, bc, e abc.
- Le sottostringhe di abc sono:

$$\varepsilon$$

$$a \quad ab \quad abc$$

$$b \quad bc$$

$$c$$

- Un prefisso, un suffisso o una sottostringa di una stringa, quando non sono la stringa stessa, sono detti propri.

Esercizio: Sia data una stringa w di lunghezza $|w| = n$. Quanti sono i suoi prefissi, i suoi suffissi e le sue sottostringhe? Quanti sono i prefissi dei suoi suffissi? Quanti i suffissi dei suoi prefissi?

STRINGHE

➜ La concatenazione di due stringhe

$$v = v_1 \cdots v_n$$

e

$$w = w_1 \cdots w_m,$$

con $n, m \geq 0$, è la stringa

$$vw = v_1 \cdots v_n w_1 \cdots w_m$$

che si ottiene facendo seguire alla prima la seconda.

➜ Esempio: se $v = abc$ e $w = cbab$ allora $vw = abccbab$.

Esercizio: Dimostrare che la concatenazione è una operazione associativa che ammette come identità la stringa vuota ε.

ALFABETI E LINGUAGGI

- Un alfabeto Σ è un insieme finito di simboli.
- Un linguaggio formale (in breve, linguaggio) è un insieme di stringhe di simboli da un alfabeto Σ.
- L'insieme vuoto $\varnothing$ e l'insieme $\{\varepsilon\}$ sono due linguaggi formali di qualunque alfabeto.
- Si denota con Σ^* il linguaggio costituito da tutte le stringhe sull'alfabeto Σ.
- Dunque

$$\Sigma^* = \{\, a_1 \cdots a_n \mid n \geq 0, a_i \in \Sigma \,\}.$$

- Ad esempio:
  - se $\Sigma = \{0\}$, allora $\Sigma^* = \{\varepsilon, 0, 00, 000, \ldots\}$;
  - se $\Sigma = \{0, 1\}$, allora $\Sigma^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, \ldots\}$.

Esercizio: Si provi che se $\Sigma \neq \varnothing$, allora Σ^* è numerabile.

Definendo inoltre:

$$\begin{cases} \Sigma^0 & = \{\varepsilon\} \\ \Sigma^{n+1} & = \{\, ax \mid a \in \Sigma, x \in \Sigma^n \,\} \end{cases}$$

si dimostri che $\Sigma^* = \bigcup_{i \geq 0} \Sigma^i$.

Qual è la cardinalità di Σ^i ?

AUTOMI A STATI FINITI: DESCRIZIONE INFORMALE

Modello “meccanico”:

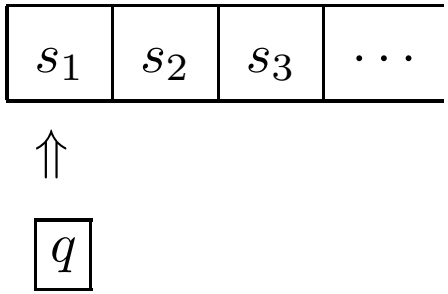
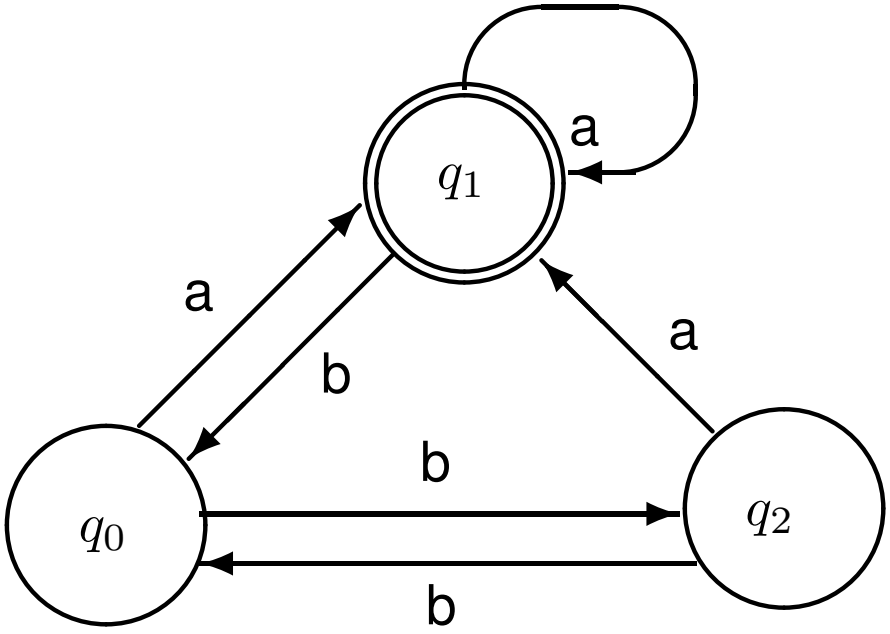


Tabella di transizione:

	a	b
q_0	q_1	q_2
q_1	q_1	q_0
q_2	q_1	q_0

AUTOMI A STATI FINITI: DESCRIZIONE INFORMALE

Rappresentazione grafica mediante grafo di transizione:



AUTOMI A STATI FINITI DETERMINISTICI: DEFINIZIONE FORMALE

Un automa a stati finiti deterministico (DFA) è una quintupla $\langle Q, \Sigma, \delta, q_0, F \rangle$ dove:

- Q è un insieme finito di stati;
- Σ è un alfabeto (alfabeto di input);
- $\delta : Q \times \Sigma \longrightarrow Q$ è la funzione di transizione;
- q_0 è lo stato iniziale;
- $F \subseteq Q$ è l'insieme degli stati finali.

Notazione: In genere, useremo p, q, r, ... per denotare stati, P,Q,R,S, ... per insiemi di stati, a,b, ... per denotare simboli di Σ, x,y,z,u,v,w, ... per denotare stringhe. Il tutto con o senza pedici (ad esempio, anche q_0, q_1, ... sono stati).

AUTOMI A STATI FINITI DETERMINISTICI: DEFINIZIONE FORMALE

Dalla funzione δ si ottiene in modo univoco $\hat{\delta} : Q \times \Sigma^* \longrightarrow Q$:

$$\begin{cases} \hat{\delta}(q, \varepsilon) & = q \\ \hat{\delta}(q, wa) & = \delta\big(\hat{\delta}(q, w), a\big) \end{cases}$$

Una stringa x è detta essere accettata da un DFA $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ se

$$\hat{\delta}(q_0, x) \in F.$$

Il linguaggio accettato da M, denotato da $L(M)$ è l'insieme delle stringhe accettate, ovvero:

$$L(M) = \left\{ x \in \Sigma^* \;\middle|\; \hat{\delta}(q_0, x) \in F \right\}$$

Un linguaggio L è detto regolare se è accettato da qualche DFA, ovvero se esiste M tale che $L = L(M)$.

$\varnothing$ e Σ^* sono linguaggi regolari. Sia $\Sigma = \{s_1, \ldots, s_n\}$: un automa M_0 che riconosce il linguaggio $\varnothing$ (ovvero: nessuna stringa è accettata) è il seguente:

	s_1	$\ldots$	s_n
q_0	q_0	$\ldots$	q_0

ove $F = \varnothing$. Infatti, poiché $\forall x : x \notin \varnothing$, si ha che:

$$\forall x \in \Sigma^* : \hat{\delta}(q_0, x) \notin F$$

Un automa per Σ^*, è invece l'automa M_1:

	s_1	$\ldots$	s_n
q_0	q_0	$\ldots$	q_0

ove $F = \{q_0\}$. Si dimostra facilmente infatti, per induzione su $|x|$ che

$$\forall x \in \Sigma^* : \hat{\delta}(q_0, x) = q_0$$

AUTOMI A STATI FINITI NON-DETERMINISTICI

Un automa a stati finiti non-deterministico (NFA) è una quintupla $\langle Q, \Sigma, \delta, q_0, F \rangle$ dove

- Q, Σ, q_0 e $F \subseteq Q$ sono come per i DFA;
- la funzione di transizione è ora del tipo

$$\delta \colon Q \times \Sigma \to \wp(Q).$$

- In particolare, possiamo avere $\delta(q, a) = \varnothing$ per qualche $q \in Q$ ed $a \in \Sigma$.
- Anche per gli NFA dalla funzione δ si ottiene in modo univoco la funzione $\hat{\delta} \colon Q \times \Sigma^* \to \wp(Q)$:

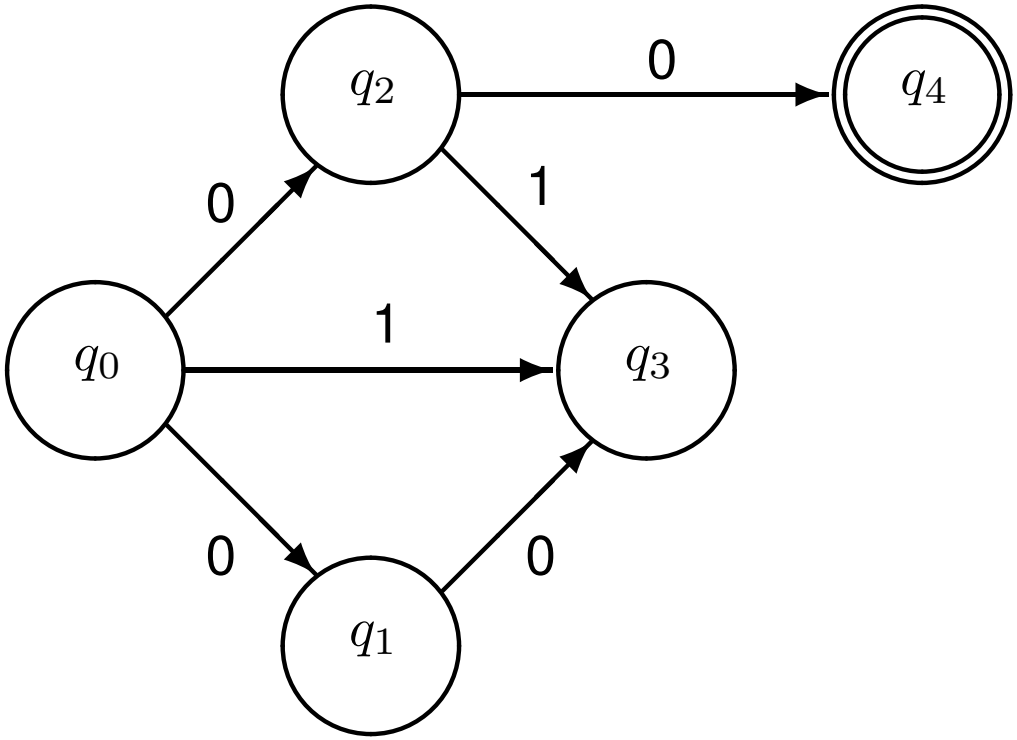
$$\begin{cases} \hat{\delta}(q, \varepsilon) & = \{q\} \\ \hat{\delta}(q, wa) & = \bigcup_{p \in \hat{\delta}(q,w)} \delta(p, a) \end{cases}$$

AUTOMI A STATI FINITI NON-DETERMINISTICI: ESEMPIO

- Una stringa x è accettata da un NFA $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ se $\hat{\delta}(q_0, x) \cap F \neq \varnothing$.
- Il linguaggio accettato da M è l'insieme delle stringhe accettate, ovvero:

$$L(M) = \left\{ x \in \Sigma^* \;\middle|\; \hat{\delta}(q_0, x) \cap F \neq \varnothing \right\}$$

AUTOMI A STATI FINITI NON-DETERMINISTICI: ESEMPIO



EQUIVALENZA TRA DFA E NFA

- ➜ Un DFA è banalmente convertibile in un NFA.
- ➜ Il teorema di Rabin-Scott (1959) mostra che un NFA è sempre convertibile in un DFA.

Teorema. *Sia* $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ *un NFA. Allora esiste un DFA* M' *tale che* $L(M) = L(M')$.

EQUIVALENZA TRA DFA E NFA: TEOREMA DI RABIN-SCOTT

Dimostrazione: Si definisca $M' = \langle Q', \Sigma', \delta', q_0', F' \rangle$ come segue:

- $\Sigma' = \Sigma$;
- $Q' = \wp(Q)$;
- $q_0' = \{q_0\}$;
- $F' = \{P \subseteq Q : P \cap F \neq \varnothing\}$;
- $\delta'(P, a) = \bigcup_{p \in P} \delta(p, a)$, per $P \in \wp(Q)$.

EQUIVALENZA TRA DFA E NFA: TEOREMA DI RABIN-SCOTT

Mostriamo per induzione sulla lunghezza della stringa di input x che

$$\hat{\delta}(q_0, x) = \hat{\delta}'(q_0', x)$$

Base: Per $|x| = 0$ il risultato è banale, poiché $q_0' = \{q_0\}$ e $x = \varepsilon$.

Passo: Supponiamo che l'ipotesi induttiva valga per tutte le stringhe x tali che $|x| \leq m$. Sia xa una stringa di lunghezza $m + 1$. Allora:

$$\begin{aligned}
\hat{\delta}'(q_0', xa) &= \delta'(\hat{\delta}'(q_0', x), a) && \text{[per def. di } \hat{\cdot} \text{ nei DFA]} \\
&= \delta'(\hat{\delta}(q_0, x), a) && \text{[per ip. ind.]} \\
&= \bigcup_{p \in \hat{\delta}(q_0, x)} \delta(p, a) && \text{[per def. di } \delta' \text{]} \\
&= \hat{\delta}(q_0, xa) && \text{[per def. di “}\hat{\cdot}\text{” negli NFA]}
\end{aligned}$$

EQUIVALENZA TRA DFA E NFA: TEOREMA DI RABIN-SCOTT

Il teorema segue dal fatto che:

$$
\begin{aligned}
x \in L(M) &\iff \hat{\delta}(q_0, x) \cap F \neq \varnothing && \text{[def. di linguaggio NFA]} \\
&\iff \hat{\delta}'(q_0', x) \cap F \neq \varnothing && \text{[proprietà sopra]} \\
&\iff \hat{\delta}'(q_0', x) \in F' && \text{[per def. di } F'\text{]} \\
&\iff x \in L(M') && \text{[per def. di linguaggio DFA]}
\end{aligned}
$$

EQUIVALENZA TRA DFA E NFA: ESERCIZI

Esercizio: Si determini il DFA equivalente al NFA:

	0	1
q_0	$\{q_0\}$	$\{q_0, q_1\}$
q_1	$\{q_1\}$	$\{q_0, q_2\}$
q_2	$\{q_1, q_2\}$	$\{q_0, q_1, q_2\}$

ove $F = \{q_2\}$. Qual è il linguaggio accettato?

Esercizio: Si descriva un NFA a 4 stati che riconosce il linguaggio delle stringhe di 0 e 1 con terzultimo elemento a 0. Si passi poi al DFA equivalente e lo si confronti con quello ottenuto nell'esercizio sopra.

AUTOMI CON ε-TRANSIZIONI

Un NFA con ε-transizioni è una quintupla $\langle Q, \Sigma, \delta, q_0, F \rangle$:

- Q, Σ, q_0 e $F \subseteq Q$ sono come per gli NFA;
- $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \longrightarrow \wp(Q)$.
- L'idea è che da uno stato è permesso passare ad un altro stato anche senza "leggere" caratteri di input.
- La definizione di $\hat{\delta} : Q \times \Sigma^* \longrightarrow \wp(Q)$ fa riferimento alle funzioni ausiliarie ε-step$\colon \wp(Q) \to \wp(Q) \ldots$

$$\varepsilon\text{-step}(S) = \left\{ q \in Q \;\middle|\; \exists p \in S \,.\, q \in \delta(p, \varepsilon) \right\};$$

$$\varepsilon\text{-step}^0(S) = S;$$

$$\varepsilon\text{-step}^{n+1}(S) = \varepsilon\text{-step}\big(\varepsilon\text{-step}^n(S)\big);$$

AUTOMI CON ε-TRANSIZIONI (CONT.)

➜ ... e ε-closure$\colon Q \to \wp(Q)$ e ε-closure$\colon \wp(Q) \to \wp(Q)$:

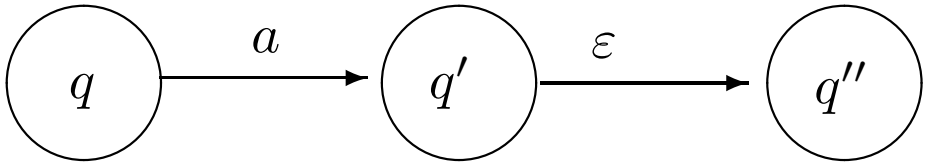
$$\varepsilon\text{-closure}(S) = \bigcup_{i \in \mathbb{N}} \varepsilon\text{-step}^i(S);$$

$$\varepsilon\text{-closure}(p) = \varepsilon\text{-closure}\big(\{p\}\big).$$

➜ $\hat{\delta}$ si può ora definire nel modo seguente:

$$\begin{cases} \hat{\delta}(q, \varepsilon) & = \varepsilon\text{-closure}(q), \\ \hat{\delta}(q, wa) & = \bigcup_{p \in \hat{\delta}(q,w)} \varepsilon\text{-closure}(\delta(p, a)). \end{cases}$$

AUTOMI CON ε-TRANSIZIONI (CONT.)



$$\delta(q,a) = \{q'\} \neq \{q', q''\} = \bigcup_{p \in \hat{\delta}(q,\varepsilon)} \varepsilon\text{-closure}(\delta(p,a)) = \hat{\delta}(q,a)$$

AUTOMI CON ε-TRANSIZIONI (CONT.)

- Il linguaggio accettato da un ε-NFA è definito come

$$L(M) = \{ x \in \Sigma^* \mid \hat{\delta}(q_0, x) \cap F \neq \varnothing \}.$$

- Osservazione: per gli ε-NFA si potrebbe assumere che l'insieme F abbia esattamente un elemento.
- Osservazione: $\hat{\delta}(q, x) = \varepsilon$-closure$(\hat{\delta}(q, x))$.

EQUIVALENZA DI ε-NFA E NFA

➜ Ogni NFA è, per definizione, un caso particolare di un ε-NFA.

➜ Ma gli ε-NFA non sono più potenti degli NFA:

Teorema. *Sia $M = \langle Q, \Sigma, \delta, q_0, F\rangle$ un ε-NFA. Allora esiste un NFA M' tale che $L(M) = L(M')$.*

Dimostrazione: Definisco $M' = \langle Q', \Sigma', \delta', q_0', F'\rangle$ come segue:

- $Q' = Q$,
- $\Sigma' = \Sigma$,
- $q_0' = q_0$,
- $F' = \begin{cases} F \cup \{q_0\}, & \text{se } \varepsilon\text{-closure}(q_0) \cap F \neq \varnothing, \\ F, & \text{altrimenti,} \end{cases}$
- $\delta'(q, a) = \hat{\delta}(q, a)$.

Equivalenza di ε-NFA e NFA

- Dobbiamo mostrare che $\hat{\delta}(q_0, x) \cap F \neq \varnothing \iff \hat{\delta}'(q_0', x) \cap F' \neq \varnothing$.
- Trattiamo distintamente i casi $x = \varepsilon$ e $x \neq \varepsilon$.

Se $x = \varepsilon$:

- $\hat{\delta}(q_0, \varepsilon) = \varepsilon$-closure$(q_0)$ e $\hat{\delta}'(q_0', \varepsilon) = \{q_0\}$, per definizione.
- $(\Longrightarrow)$ Se ε-closure$(q_0) \cap F \neq \varnothing$ allora, per def. di F' vale che $q_0 \in F'$; dunque $\{q_0\} \cap F' \neq \varnothing$.
- $(\Longleftarrow)$ Sia ora $\{q_0\} \cap F' \neq \varnothing$. Allora $q_0 \in F'$. Due casi sono possibili:
  1. se $q_0 \in F$ allora ε-closure$(q_0) \cap F \neq \varnothing$ (in quanto $q_0 \in \varepsilon$-closure(q_0)).
  2. Altrimenti, se $q_0 \in F' \setminus F$ per definizione di F' si ha che ε-closure$(q_0) \cap F \neq \varnothing$.

EQUIVALENZA DI ε-NFA E NFA (CONT.)

- Dobbiamo mostrare che $\hat{\delta}(q_0, x) \cap F \neq \varnothing \iff \hat{\delta}'(q_0', x) \cap F' \neq \varnothing$.

Se $x \neq \varepsilon$:

- Mostriamo, per induzione su $|x| \geq 1$, che $\hat{\delta}'(q_0', x) = \hat{\delta}(q_0, x)$:
- *Base:* $|x| = 1$. Allora $x = a$ per qualche simbolo $a \in \Sigma$. Ma allora $\hat{\delta}'(q_0', a) = \delta'(q_0', a) = \hat{\delta}(q_0, a)$ per definizione.

EQUIVALENZA DI ε-NFA E NFA (CONT.)

➜ *Passo:* Assumiamo che la tesi valga per tutte le stringhe x tali che $1 \leq |x| \leq m$. Sia xa una stringa di lunghezza $m+1$. Allora:

$$\begin{aligned}
\hat{\delta}'(q_0', xa) &= \bigcup_{p \in \hat{\delta}'(q_0', x)} \delta'(p, a) && [\text{def. di } \hat{\delta}'] \\
&= \bigcup_{p \in \hat{\delta}'(q_0', x)} \hat{\delta}(p, a) && [\text{def. di } \delta'] \\
&= \bigcup_{p \in \hat{\delta}(q_0, x)} \hat{\delta}(p, a) && [\text{ip. ind.}] \\
&= \bigcup_{p \in \hat{\delta}(q_0, x)} \bigcup_{r \in \hat{\delta}(p, \varepsilon)} \varepsilon\text{-closure}(\delta(r, a)) && [\text{def. di } \hat{\delta}] \\
&= \bigcup_{p \in \hat{\delta}(q_0, x)} \varepsilon\text{-closure}(\delta(p, a)) && [\hat{\delta}(q_0, x) \text{ chiuso per } \varepsilon\text{-closure}] \\
&= \hat{\delta}(q_0, xa) && [\text{def. di } \hat{\delta}]
\end{aligned}$$

EQUIVALENZA DI ε-NFA E NFA (CONT.)

➜ Dobbiamo mostrare che $\hat{\delta}(q_0, x) \cap F \neq \varnothing \iff \hat{\delta}'(q_0', x) \cap F' \neq \varnothing$.

Se $x \neq \varepsilon$:

➜ Poiché $F \subseteq F'$ e $\hat{\delta}'(q_0', x) = \hat{\delta}(q_0, x)$ l'implicazione ($\Longrightarrow$) deriva immediatamente.

➜ Per l'altra implicazione, l'unico problema si avrebbe nel caso:

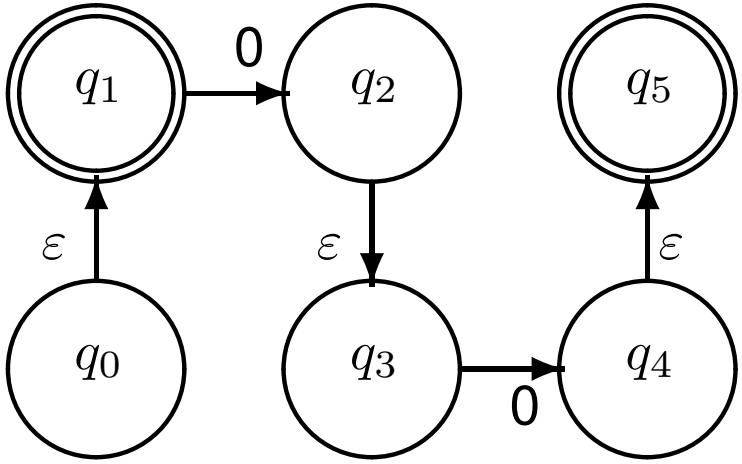
1. $q_0 \in \hat{\delta}'(q_0', x)$,
2. $q_0' \in F'$,
3. $q_0 \notin F$.

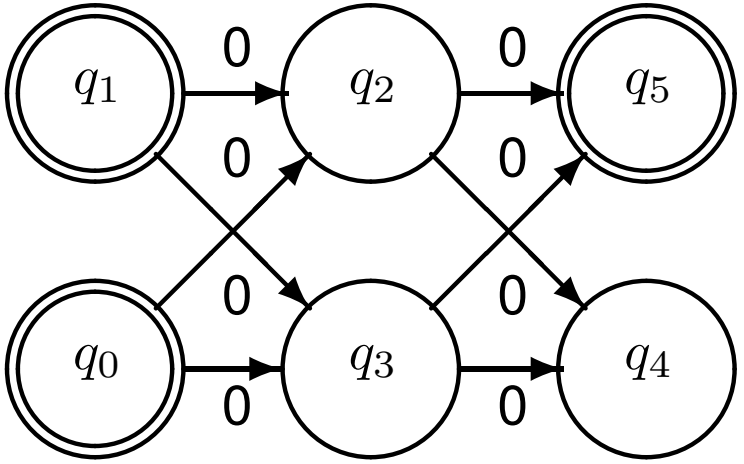
Poiché $\hat{\delta}'(q_0', x) = \hat{\delta}(q_0, x)$, si ha che $q_0 \in \hat{\delta}(q_0, x)$. Ma, per definizione di $\hat{\delta}$, anche ogni elemento della sua ε-closure appartiene a $\hat{\delta}(q_0, x)$. Tale ε-closure, poiché $q_0 \in F'$, interseca F.

Equivalenza di ε-NFA e NFA (cont.)

Corollario 1. *Le classi di linguaggi riconosciute da DFA, NFA e ε-NFA coincidono (linguaggi regolari).*

EQUIVALENZA DI ε-NFA E NFA (ESEMPIO)





- Esempio della trasformazione da ε-NFA in NFA.
- Il numero degli stati rimane costante, ma q_0 viene aggiunto agli stati finali.

RAPPRESENTAZIONE DEI LINGUAGGI

- Il problema della rappresentazione finita di linguaggi infiniti è affrontabile da tre distinti punti di vista:
  1. riconoscitivo-analitico, quello che già conosciamo, in cui il linguaggio è visto come l'insieme delle stringhe riconosciute o accettate da strutture finite dette automi;
  2. generativo-sintetico, in cui il linguaggio è visto come l'insieme delle stringhe generate da strutture finite dette grammatiche;
  3. algebrico, in cui il linguaggio è rappresentato da un'espressione algebrica o è la soluzione di un sistema di relazioni algebriche.
- Le espressioni regolari, che ora vedremo, ricadono nell'approccio di tipo algebrico.

OPERAZIONI SUI LINGUAGGI

Sia Σ un alfabeto e L, L_1, L_2 insiemi di stringhe di Σ^*.

- La concatenazione di L_1 e L_2 è l'insieme:

$$L_1 L_2 = \{\, xy \in \Sigma^* \mid x \in L_1, y \in L_2 \,\}.$$

- La chiusura (di Kleene) di L, è l'insieme:

$$L^* = \bigcup_{i \geq 0} L^i,$$

dove

$$L^0 = \{\varepsilon\},$$
$$L^{i+1} = LL^i.$$

OPERAZIONI SUI LINGUAGGI (CONT.)

- La *chiusura positiva di* L è l'insieme

$$L^+ = \bigcup_{i \geq 1} L^i.$$

- Nota: $L^+ = LL^*$.
- Altre operazioni sui linguaggi che menzioneremo nel seguito sono le normali operazioni insiemistiche di unione, intersezione e complemento rispetto a Σ^*.

ESPRESSIONI REGOLARI

Sia Σ un alfabeto. Le espressioni regolari su Σ e gli *insiemi* che esse denotano sono definiti ricorsivamente nel modo seguente:

1. $\varnothing$ è una espressione regolare che denota l'insieme vuoto.
2. ε è una espressione regolare che denota l'insieme $\{\varepsilon\}$.
3. Per ogni simbolo $a \in \Sigma$, a è una espressione regolare che denota l'insieme $\{a\}$.
4. Se r e s sono espressioni regolari denotanti rispettivamente gli insiemi R ed S, allora $(r + s)$, (rs), e (r^*) sono espressioni regolari che denotano gli insiemi $R \cup S$, RS, e R^* rispettivamente.

Se r è una espressione regolare, indicheremo con $L(r)$ il linguaggio denotato da r.

ESPRESSIONI REGOLARI (CONT.)

Possiamo dare una definizione più formale del linguaggio denotato da un'espressione regolare: se r ed s sono espressioni regolari,

$$
\begin{aligned}
L(\varnothing) &= \varnothing, \\
L(\varepsilon) &= \{\varepsilon\}, \\
L(a) &= \{a\}, \text{ per ogni } a \in \Sigma, \\
L(r+s) &= L(r) \cup L(s), \\
L(rs) &= L(r)L(s), \\
L(r^*) &= L(r)^*.
\end{aligned}
$$

ESPRESSIONI REGOLARI (ESEMPI)

- Sia $\Sigma = \{0, 1\}$. Il linguaggio associato all'espressione regolare $r = 1(0 + 1)^*00$ è

$$L(r) = L(1)(L(0) \cup L(1))^* L(0)L(0) = \{\, 1x00 \mid x \in \Sigma^* \,\},$$

  ovvero l'insieme delle stringhe binarie rappresentanti un multiplo di 4.

- Sia Σ l'insieme dei caratteri ASCII. Una stringa w è un identificatore C o C++ se e solo se $w \in L(r)$ con

$$r = (\texttt{A}+\cdots+\texttt{Z}+\texttt{a}+\cdots+\texttt{z}+\texttt{\_})(\texttt{A}+\cdots+\texttt{Z}+\texttt{a}+\cdots+\texttt{z}+\texttt{\_}+\texttt{0}+\cdots+\texttt{9})^*,$$

  ovvero w è una sequenza non vuota di lettere, cifre decimali o underscore in cui il primo carattere non è una cifra decimale.

EQUIVALENZA TRA DFA ED ESPRESSIONI REGOLARI

Teorema (McNaughton & Yamada, 1960)**.** *Sia r una espressione regolare. Allora esiste un ε-NFA M tale che $L(M) = L(r)$.*

Dimostrazione: Costruiremo un ε-NFA siffatto, con un unico stato finale, per induzione sulla struttura dell'espressione regolare r.

Base: Ci sono tre casi base:

- l'automa:

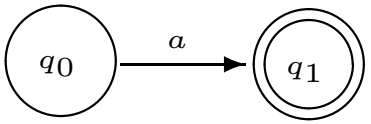
  riconosce il linguaggio $\{\varepsilon\}$;

EQUIVALENZA TRA DFA ED ESPRESSIONI REGOLARI (CONT.)

- l'automa

  riconosce il linguaggio $\varnothing$;

- l'automa

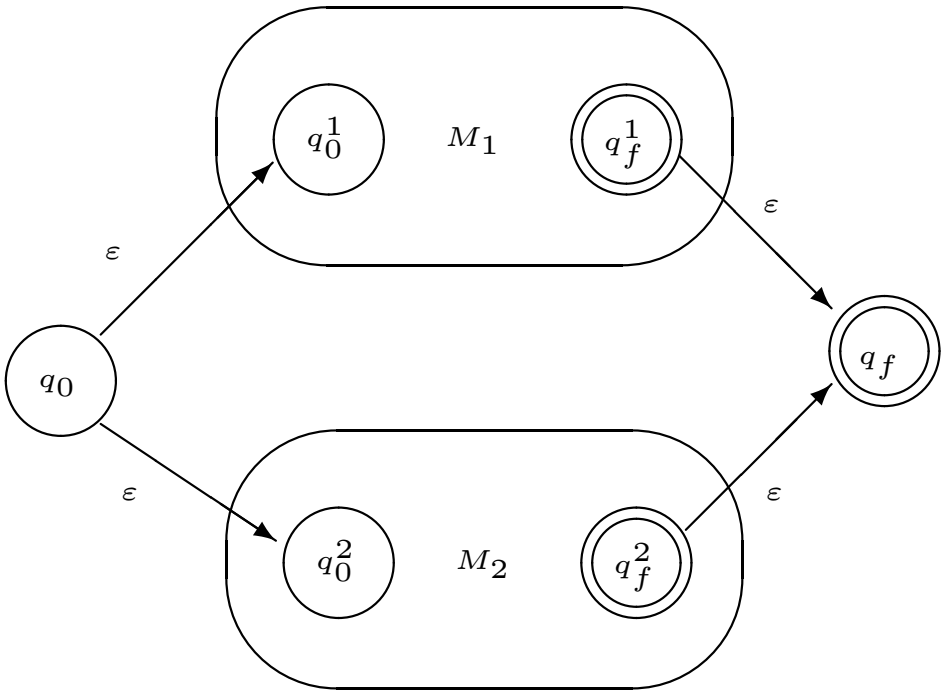


  riconosce il linguaggio $\{a\}$.

EQUIVALENZA TRA DFA ED ESPRESSIONI REGOLARI (CONT.)

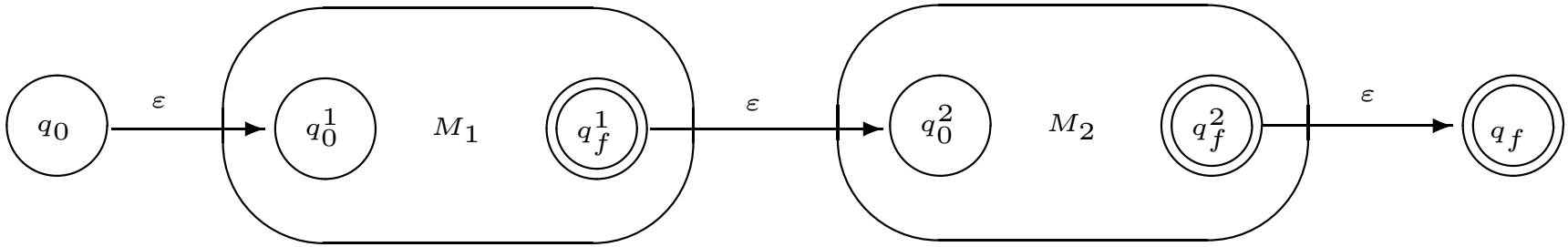
Passo: Anche qui abbiamo tre casi da analizzare:

- $r = r_1 + r_2$. Per $i = 1, 2$, sia M_i, con stato iniziale q_0^i e stato finale q_f^i l'automa che riconosce $L(r_i)$. Il seguente automa, con stato iniziale q_0 e stato finale q_f riconosce il linguaggio $L(r) = L(r_1) \cup L(r_2)$:



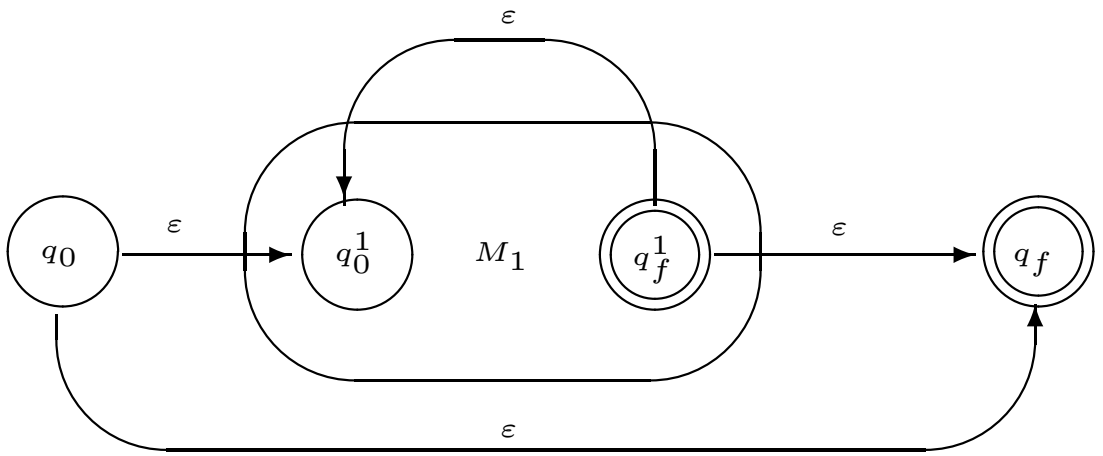
EQUIVALENZA TRA DFA ED ESPRESSIONI REGOLARI (CONT.)

➜ $r = r_1 r_2$. Per $i = 1, 2$, sia M_i, con stato iniziale q_0^i e stato finale q_f^i l'automa che riconosce $L(r_i)$. L'esistenza di tali automi è assicurata dall'ipotesi induttiva. Il seguente automa, con stato iniziale q_0 e stato finale q_f riconosce il linguaggio $L(r_1)L(r_2)$:



EQUIVALENZA TRA DFA ED ESPRESSIONI REGOLARI (CONT.)

➜ $r = r_1^*$. Sia M_1, con stato iniziale q_0^1 e stato finale q_f^1 l'automa che riconosce $L(r_1)$. L'esistenza di tale automa è assicurata dall'ipotesi induttiva. Il seguente automa, con stato iniziale q_0 e stato finale q_f riconosce $L(r) = (L(r_1))^*$:



Le dimostrazioni che tali automi riconoscono esattamente i linguaggi a loro assegnati è lasciata per esercizio.

EQUIVALENZA TRA DFA ED ESPRESSIONI REGOLARI (CONT.)

Teorema. *Sia M un DFA. Allora esiste una espressione regolare r tale che $L(M) = L(r)$.*

Dimostrazione: Sia $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ con $Q = \{1, 2, \ldots, n\}$ e $q_0 = 1$.

Con $r_{ij}^{(k)}$ denotiamo l'espressione regolare che definisce l'insieme di stringhe che permettono di spostarsi dallo stato i allo stato j nel DFA M, senza passare attraverso stati maggiori di k (salvo i e j).

Definiamo induttivamente $r_{ij}^{(k)}$:

Base: con $k = 0$ ci sono solo due possibilità:

1. una transizione dallo stato i allo stato j;
2. un cammino di lunghezza 0 e $i = j$.

EQUIVALENZA TRA DFA ED ESPRESSIONI REGOLARI (CONT.)

Se $i \neq j$ allora è possibile solo il primo caso, quindi:

i. se non c'è nessuna transizione da i a j, allora $r_{ij}^{(0)} = \varnothing$;

ii. se c'è esattamente una transizione da i a j per il simbolo a, allora $r_{ij}^{(0)} = a$;

iii. se ci sono $p > 1$ simboli $a_1, a_2, \ldots, a_p$ corrispondenti ad altrettante transizioni da i a j, allora $r_{ij}^{(0)} = a_1 + a_2 + \cdots + a_p$.

Se $i = j$, allora è possibile avere un cammino di lunghezza 0. Le definizioni vanno dunque così modificate:

i. $r_{ij}^{(0)} = \varepsilon$ (nessun simbolo a);

ii. $r_{ij}^{(0)} = \varepsilon + a$ (un solo simbolo a);

iii. $r_{ij}^{(0)} = \varepsilon + a_1 + a_2 + \cdots + a_p$ (p simboli).

EQUIVALENZA TRA DFA ED ESPRESSIONI REGOLARI (CONT.)

Passo: Sia $k > 0$. Per ogni coppia di stati i e j, consideriamo tutti i cammini da i a j che attraversano stati non maggiori di k (possono essere infiniti). Per ognuno di questi ci sono due casi:

a. Il cammino non passa attraverso lo stato k: dunque, per ipotesi induttiva, la stringa corrispondente è in $r_{ij}^{(k-1)}$;

b. Il cammino passa attraverso lo stato k almeno una volta. In questo caso spezzeremo il cammino in tre parti:

   1. il primo andrà dallo stato i allo stato k senza passare attraverso k, e corrisponderà per i. i. ad una stringa in $r_{ik}^{(k-1)}$;
   2. l'ultimo andrà dallo stato k allo stato j senza passare attraverso k, e corrisponderà per i. i. ad una stringa in $r_{kj}^{(k-1)}$;

EQUIVALENZA TRA DFA ED ESPRESSIONI REGOLARI (CONT.)

3. la parte centrale sarà a sua volta la concatenazione di zero o più frammenti che vanno da k a k *senza* passare da k. Ognuno di questi frammenti corrisponde per i. i. ad una stringa in $r_{kk}^{(k-1)}$. La parte centrale corrisponderà quindi ad una stringa in $\left(r_{kk}^{(k-1)}\right)^*$.

Combinando i casi a. e b., otteniamo

$$r_{ij}^{(k)} = r_{ij}^{(k-1)} + r_{ik}^{(k-1)} \left(r_{kk}^{(k-1)}\right)^* r_{kj}^{(k-1)}.$$

Se $F = \{f_1, \ldots, f_h\}$, l'espressione cercata è

$$r = r_{1f_1}^{(n)} + \cdots + r_{1f_h}^{(n)}.$$

LINGUAGGI REGOLARI E NON

- Abbiamo visto diversi metodi per stabilire se un dato linguaggio è regolare (esibizione di un DFA, NFA, ε-NFA o di una espressione regolare).
- Ci chiediamo ora: tutti i linguaggi sono regolari?
- In caso contrario: come si può dimostrare che un linguaggio non è regolare?
- (Non essere riusciti a trovare un automa o un'espressione regolare adeguati, chiaramente, non dimostra nulla).

IL "PUMPING LEMMA" PER LINGUAGGI REGOLARI

Lemma 1 (Bar-Hillel, Perles, Shamir, 1961)**.** *Sia L un linguaggio regolare. Allora esiste una costante $n \in \mathbb{N}$ tale che, per ogni $z \in L$ tale che $|z| \geq n$, esistono tre stringhe u, v, w tali che:*

1. $z = uvw$,
2. $|uv| \leq n$,
3. $|v| > 0$, *e*
4. *per ogni $i \geq 0$ vale che $uv^i w \in L$.*

IL "PUMPING LEMMA" PER LINGUAGGI REGOLARI (CONT.)

Dimostrazione:

- Sia $M = \langle \{q_0, \ldots, q_{n-1}\}, \Sigma, \delta, q_0, F \rangle$ un DFA tale che $L = L(M)$.
- Sia $z = a_1 \cdots a_m \in L$ arbitraria con $m \geq n$ (se una tale z non esistesse, il lemma varrebbe banalmente).
- Per $i = 1, \ldots, n$ (ricordando che $n \leq m$) si consideri l'evoluzione degli stati $\hat{\delta}(q_0, a_1 \cdots a_i)$:

$$\underbrace{q_0, \hat{\delta}(q_0, a_1), \hat{\delta}(q_0, a_1 a_2), \ldots, \hat{\delta}(q_0, a_1 \cdots a_n)}_{n+1}$$

- Poiché gli stati dell'automa sono n per ipotesi, esiste (almeno) uno stato $\bar{q}$ raggiunto (almeno) due volte:

$$\begin{aligned} \hat{\delta}(q_0, a_1 \cdots a_{i_1}) &= \bar{q} \\ &= \hat{\delta}(q_0, a_1 \cdots a_{i_1} \cdots a_{i_2}) \end{aligned}$$

con $i_2 > i_1$.

IL "PUMPING LEMMA" PER LINGUAGGI REGOLARI (CONT.)

- Si pongono

$$u = a_1 \cdots a_{i_1},$$

$$v = a_{i_1+1} \cdots a_{i_2},$$

$$w = a_{i_2+1} \cdots a_m.$$

- Abbiamo $|uv| \leq n$ e $|v| > 0$ per costruzione.
- Sappiamo che $\hat{\delta}(q_0, u) = \bar{q}$ e $\hat{\delta}(q_0, uv) = \bar{q}$. Usando l'esercizio 3.8 si ha che $\hat{\delta}(\bar{q}, v) = \bar{q}$.
- Per ipotesi abbiamo che $\hat{\delta}(q_0, uvw) = q'$ per qualche $q' \in F$.
- Siccome $\hat{\delta}(q_0, uv) = \bar{q}$, usando ancora l'esercizio 3.8 si ha che $\hat{\delta}(\bar{q}, w) = q'$.
- Per induzione su $i \geq 0$ si mostra (farlo per esercizio) che $\hat{\delta}(q_0, uv^i) = \bar{q}$.
- Si può quindi concludere che $\hat{\delta}(q_0, uv^i w) = q' \in F$, come volevasi.

IL "PUMPING LEMMA" PER LINGUAGGI REGOLARI (CONT.)

Corollario 2. *La costante $n \in \mathbb{N}$ del "Pumping Lemma" può essere presa come il minimo numero di stati degli automi che riconoscono L.*

Dimostrazione: Immediata dalla dimostrazione del Lemma.

IL "PUMPING LEMMA" PER LINGUAGGI REGOLARI (OSSERVAZIONE)

- Il lemma asserisce, dato un linguaggio regolare L, la veridicità della formula

$$\exists n \in \mathbb{N} \,.\, \forall z \in L : \Big(|z| \geq n \rightarrow \exists u, v, w \,.\, \big(z = uvw \wedge |uv| \leq n \wedge |v| > 0 \wedge \forall i \in \mathbb{N} : uv^i w \in L \big) \Big)$$

- Il "Pumping Lemma" può essere usato per mostrare che un dato linguaggio non è regolare (qualora non lo sia!). Per fare ciò, bisogna mostrare che vale la negazione della formula di cui sopra, ovvero di

$$\forall n \in \mathbb{N} : \exists z \in L \,.\, \Big(|z| \geq n \wedge \forall u, v, w \,.\, \Big(\big(z = uvw \wedge |uv| \leq n \wedge |v| > 0 \big) \rightarrow \exists i \in \mathbb{N} \,.\, uv^i w \notin L \Big) \Big)$$

Il "Pumping Lemma" per linguaggi regolari (esempio)

- Il linguaggio $L = \{ 0^i 1^i \mid i \geq 0 \}$ non è regolare.
- Si prenda un numero naturale n arbitrario e si scelga, tra tutte le stringhe in L di lunghezza maggiore o uguale a n, la stringa $z = 0^n 1^n \in L$.
- Bisogna ora mostrare che prese comunque u, v e w tali che $z = uvw$, $|uv| \leq n$ e $|v| > 0$ esiste almeno un $i \geq 0$ tale per cui $uv^i w \notin L$.
- Essendo $|uv| \leq n$, tali modi di suddividere z sono tutti riconducibili al seguente schema, con $a + b + c = n$ e $b > 0$:

$$u = 0^a,$$
$$v = 0^b,$$
$$w = 0^c 1^n,$$

- "Pompando" v si ottengono stringhe al di fuori del linguaggio, da cui l'assurdo. Ad esempio, con $i = 0$ vale che $uv^0 w = 0^a 0^c 1^n \notin L$, poiché $a + c < n$.

PROPRIETÀ DI CHIUSURA DEI LINGUAGGI REGOLARI

Teorema. *I linguaggi regolari sono chiusi rispetto alle operazioni di unione, concatenazione e chiusura di Kleene.*

Dimostrazione: Immediata usando i risultati visti sulle espressioni regolari.

Teorema. *I linguaggi regolari sono chiusi rispetto alla operazione di complementazione. Ovvero, se $L \subseteq \Sigma^*$ è regolare, anche $\bar{L} = \Sigma^* \setminus L$ è regolare.*

Dimostrazione: Sia $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ il DFA che riconosce L. Allora, banalmente, $M' = \langle Q, \Sigma, \delta, q_0, Q \setminus F \rangle$ riconosce $\bar{L}$.

Corollario 3. *I linguaggi regolari sono chiusi rispetto all'intersezione.*

Dimostrazione: Immediata dal fatto che $L_1 \cap L_2 = \overline{(\bar{L}_1 \cup \bar{L}_2)}$.

RISULTATI DI DECIDIBILITÀ

Teorema (Appartenenza)**.** *Il problema dell'appartenenza per linguaggi regolari (data una descrizione del linguaggio regolare L [mediante e.r. o automa] e una stringa x, decidere se $x \in L$ o meno) è decidibile.*

Dimostrazione: Immediata.

Teorema (Vuoto-infinito)**.** *L'insieme delle stringhe accettate da un DFA M con n stati è:*

1. *non vuoto se e solo se accetta una stringa di lunghezza inferiore a n;*
2. *infinito se e solo se l'automa accetta una stringa di lunghezza ℓ, $n \leq \ell < 2n$.*

RISULTATI DI DECIDIBILITÀ (CONT.)

Dimostrazione di 1:

- ($\Longleftarrow$) Se M accetta una stringa (di lunghezza inferiore a n) allora L è non vuoto.
- ($\Longrightarrow$) Supponiamo $L \neq \varnothing$. Allora M accetta almeno una stringa z, $|z| = m$.
  - Se $m < n$, la tesi è provata.
  - Altrimenti, per il pumping lemma, $z = uvw$ con $|v| \geq 1$, e $uv^0w = uw$ è accettata da M.
    - Se $|uw| < n$ la tesi è provata.
    - Altrimenti si proceda iterativamente ripartendo con $z = uw$ (che ha lunghezza strettamente minore a m)...
  - ...dopo al più $m - n$ iterazioni si otterrà una stringa di lunghezza inferiore a n.

RISULTATI DI DECIDIBILITÀ (CONT.)

Dimostrazione di 2:

- ($\Longleftarrow$) Supponiamo M accetti una stringa z di lunghezza ℓ con $n \leq \ell < 2n$.
  - Per il pumping lemma, $z = uvw$, $|v| \geq 1$ e $\{\, uv^i w \mid i \in \mathbb{N} \,\} \subseteq L(M)$.
  - $\{\, uv^i w \mid i \in \mathbb{N} \,\}$ è un insieme infinito, come volevasi.
- ($\Longrightarrow$) Sia $L(M)$ infinito.
  - Allora esiste $z \in L(M)$ tale che $|z| = m \geq 2n$.
  - Per il pumping lemma $z = uvw$ con $|uv| \leq n$, $|v| \geq 1$ (dunque $|uw| \geq n$) e $z' = uw \in L(M)$.
    - Se $|z'| < 2n$, allora la tesi è dimostrata.
    - Altrimenti, si reiteri il procedimento partendo dalla stringa $z' = uw$ (più corta di z)...
  - ...in un numero finito di passi si trova la stringa cercata.

RISULTATI DI DECIDIBILITÀ (CONT.)

Corollario 4. *Sia M DFA. Allora i problemi $L(M) = \varnothing$ e $L(M)$ è infinito sono entrambi decidibili.*

Teorema (Equivalenza)**.** *Dati due DFA M_1 e M_2, il problema di stabilire se $L(M_1) = L(M_2)$ è decidibile.*

Dimostrazione: $L(M_1) = L(M_2)$ è insiemisticamente equivalente a

$$\big(L(M_1) \cap \overline{L(M_2)}\big) \cup \big(\overline{L(M_1)} \cap L(M_2)\big) = \varnothing.$$

Sappiamo che esiste un automa M_3 tale che

$$L(M_3) = \big(L(M_1) \cap \overline{L(M_2)}\big) \cup \big(\overline{L(M_1)} \cap L(M_2)\big).$$

Il risultato segue dalla decidibilità del problema del vuoto.

Relazioni e classi di equivalenza

- Dato un insieme S, una relazione di equivalenza $R \subseteq S \times S$ (univocamente, a meno di ridenominazione degli indici) una partizione di $S = \bigcup_{i \in I} S_i$ dove, per ogni $i, j \in I$ con $i \neq j$, si ha che:
  1. $S_i \neq \varnothing$;
  2. $S_i \cap S_j = \varnothing$;
  3. $\forall a, b \in S_i : a \; R \; b$;
  4. $\forall a \in S_i : \forall b \in S_j : \neg(a \; R \; b)$.
- Le S_i sono dette classi di equivalenza.
- Se $a \in S_i$ allora con la notazione $[a]_R$ (o semplicemente $[a]$ quando R è chiara dal contesto) si denota la classe S_i.
- Se I è un insieme finito R si dice di indice finito ($|I|$) su S.
- Date due relazioni di equivalenza R_1 e R_2 sullo stesso insieme S, R_1 è un raffinamento di R_2 se ogni classe di equivalenza della partizione indotta da R_1 è sottoinsieme di qualche classe di equivalenza della partizione indotta da R_2.

RELAZIONI E CLASSI DI EQUIVALENZA (ESEMPIO)

- Sia $S = \{2, 3, 4, 5\}$.
- Siano

$$P_1 = \big\{\{2, 3, 5\}, \{4\}\big\}$$
$$P_2 = \big\{\{2\}, \{3, 5\}, \{4\}\big\}$$

le partizioni di S ottenute a partire dalle relazioni di equivalenza

$$R_1 = \big\{\, (x, y) \;\big|\; x \text{ e } y \text{ sono entrambi primi o uguali tra loro} \,\big\},$$
$$R_2 = \big\{\, (x, y) \;\big|\; x \text{ e } y \text{ sono entrambi primi e dispari, o uguali tra loro} \,\big\}.$$

- R_2 è un raffinamento di R_1.

DUE RELAZIONI INDOTTE DA LINGUAGGI E AUTOMI

- Sia $L \subseteq \Sigma^*$ un linguaggio qualsiasi.
- La relazione $R_L \subseteq \Sigma^* \times \Sigma^*$ è definita da

$$R_L = \{ (x, y) \mid \forall z \in \Sigma^* : xz \in L \leftrightarrow yz \in L \}.$$

- Sia $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ un DFA.
- La relazione $R_M \subseteq \Sigma^* \times \Sigma^*$ è definita da

$$R_M = \{ (x, y) \mid \hat{\delta}(q_0, x) = \hat{\delta}(q_0, y) \}.$$

- Si definisce il linguaggio

$$L_q = \{ x \in \Sigma^* \mid \hat{\delta}(q_0, x) = q \}$$

  il linguaggio associato allo stato q dell'automa.
- Le classi di equivalenza di R_M sono esattamente i linguaggi associati ad ogni stato dell'automa M.

DUE RELAZIONI INDOTTE DA LINGUAGGI E AUTOMI (CONT.)

Lemma 2. *R_L e R_M sono relazioni di equivalenza.*

Dimostrazione: Fare per esercizio.

- Una relazione $R \subseteq \Sigma^* \times \Sigma^*$ che gode della proprietà

$$x\ R\ y \quad \Longrightarrow \quad \forall z \in \Sigma^* : xz\ R\ yz$$

  si dice invariante a destra (rispetto alla concatenazione).
- Dall'esercizio 3.8 segue che R_M è invariante a destra.
- Anche R_L è invariante a destra:
  - Siano $x, y \in \Sigma^*$ tali che $x\ R_L\ y$ e $z \in \Sigma^*$,
  - Se non avessimo $xz\ R_L\ yz$ allora esisterebbe w tale che $xzw \in L$ e $yzw \notin L$ (o viceversa).
  - Ma in tal caso, con $z' = zw$, mostreremmo che $x \not R_L\ y$: assurdo.

IL TEOREMA DI MYHILL-NERODE

Teorema (Myhill-Nerode, 1957–58)**.** *I seguenti enunciati sono equivalenti:*

1. $L \subseteq \Sigma^*$ *è accettato da un qualche DFA;*
2. L *è l'unione di alcune classi di equivalenza di* Σ^* *indotte da una relazione invariante a destra e di indice finito;*
3. R_L *è di indice finito.*

IL TEOREMA DI MYHILL-NERODE (DIM. DI $(1) \Longrightarrow (2)$)

- Sia L accettato da un DFA $M = \langle Q, \Sigma, \delta, q_0, F \rangle$. Mostreremo che R_M è la relazione che soddisfa il punto (2).
- Per definizione di linguaggio riconosciuto da un automa,

$$L = \bigcup_{q \in F} \{\, x \in \Sigma^* \mid \hat{\delta}(q_0, x) = q \,\}.$$

- Siccome R_M è invariante a destra, gli insiemi

$$L_q = \{\, x \in \Sigma^* \mid \hat{\delta}(q_0, x) = q \,\}$$

costituiscono le classi di equivalenza della partizione indotta da R_M.

IL TEOREMA DI MYHILL-NERODE (DIM. DI $(2) \Longrightarrow (3)$)

- Mostreremo che ogni relazione di equivalenza R che soddisfa (2) è un raffinamento di R_L.
- Sia $x \in \Sigma^*$: vogliamo mostrare che $[x]_R \subseteq [x]_{R_L}$.
- Sia $y \in [x]_R$ (dunque $x\ R\ y$). Poiché R è invariante a destra per ipotesi, allora $xz\ R\ yz$ per ogni $z \in \Sigma^*$.
- Poiché L è unione di classi di equivalenza di R, ciò implica che ogni qualvolta vRw si ha che $v \in L$ sse $w \in L$. Pertanto per ogni $z \in \Sigma^*$, $xz \in L$ sse $yz \in L$. Ma allora $x\ R_L\ y$ per definizione, dunque $y \in [x]_{R_L}$.
- L'indice di R_L è minore o uguale di quello di R, che per ipotesi è finito.

IL TEOREMA DI MYHILL-NERODE (DIM. DI $(3) \Longrightarrow (1)$)

Si costruisce un DFA $M' = \langle Q', \Sigma', \delta', q_0', F' \rangle$ che riconosce L. Sia

- Q' l'insieme (finito per ipotesi) di classi di equivalenza di R_L,
- Σ' lo stesso di L,
- $\delta'([x], a) = [xa]$ (la definizione ha senso indipendentemente dalla scelta di x in quanto R_L è invariante a destra),
- $q_0' = [\varepsilon]$,
- $F' = \{[x] : x \in L\}$.

Si tratta di mostrare che $L(M') = L$. Si verifica per induzione su $|y| \geq 0$ che $\hat{\delta}'([x], y) = [xy]$. Pertanto si ha che:

$$\hat{\delta}'(q_0', x) = \hat{\delta}'([\varepsilon], x) = [\varepsilon x] = [x]$$

e dunque

$$x \in L(M') \quad \text{sse} \quad \hat{\delta}'(q_0', x) \in F' \quad \text{sse} \quad [x] \in F' \quad \text{sse} \quad x \in L\,.$$

MINIMIZZAZIONE DI DFA

Teorema. *Per ogni linguaggio regolare L esiste un automa M con minimo numero di stati tale che $L = L(M)$, unico a meno di isomorfismo (ovvero ridenominazione di stati).*

Dimostrazione:

- ➜ Sia L accettato da un DFA M. Σ^* è partizionato da R_M negli insiemi di linguaggi accettati dai singoli stati di M.
- ➜ Dal Teorema di Myhill-Nerode ($1 \to 2$), R_M raffina R_L e pertanto $|Q| \geq |Q'|$ ($M' = (Q', \ldots)$ è l'automa costruito come nella dimostrazione ($3 \to 1$) del Teorema).
- ➜ Dunque l'automa M' ha il minor numero di stati possibile.

MINIMIZZAZIONE DI DFA (CONT.)

- Supponiamo ora che $|Q| = |Q'|$ e definiamo $f \subseteq Q \times Q'$ in modo che

$$f(q) = [x] \qquad \Longleftrightarrow \qquad \hat{\delta}(q_0, x) = q.$$

- f è una funzione: supponiamo $f(q) = [y_1]$ e $f(q) = [y_2]$. Allora $\hat{\delta}(q_0, y_1) = q$ e $\hat{\delta}(q_0, y_2) = q$. Ma allora $y_1 \; R_M \; y_2$. Poiché R_M è raffinamento di R_L si ha anche $[y_1] = [y_2]$.
- f è suriettiva, in quanto per ogni $x \in \Sigma^*$ si ha che esiste $q = \hat{\delta}(q_0, x)$ tale che $f(q) = [x]$.
- Poiché $|Q| = |Q'|$ ciò implica che f è biiettiva e dunque è una funzione di rinomina dei nodi.

MINIMIZZAZIONE DI DFA (CONT.)

➜ Per mostrare che i due automi sono isomorfi rimane da mostrare che il seguente diagramma commuta, per ogni $q \in Q$ e per ogni $a \in \Sigma$:

$$\begin{array}{ccc} q & \xrightarrow{f} & [x] \\ \downarrow a & & \downarrow a \\ q' & \xrightarrow{f} & [xa] \end{array}$$

➜ In altri termini, dati $f(q) = [x]$, $\delta(q, a) = q'$, $\delta'([x], a) = [xa]$, dobbiamo mostrare che $f(q') = [xa]$.

➜ Infatti:

$$\begin{aligned} f(q') = [xa] &\iff q' = \hat{\delta}(q_0, xa) && \text{[def. di } f\text{]} \\ &\iff q' = \delta(\hat{\delta}(q_0, x), a) && \text{[def. di } \hat{\delta}\text{]} \\ &\iff q' = \delta(q, a) && \text{[vero per ipotesi]} \end{aligned}$$

MINIMIZZAZIONE DI DFA (CONT.)

Algoritmo di minimizzazione di un DFA:

input: $M = \langle Q, \Sigma, \delta, q_0, F \rangle$;

output: $M' = \langle Q', \Sigma, \delta', q_0', F' \rangle$ t.c. $L(M) = L(M')$ e

$(\forall M'' = \langle Q'', \Sigma, \delta'', q_0'', F'' \rangle)(L(M) = L(M'') \rightarrow |Q'| \leq |Q''|)$;

MINIMIZZAZIONE DI DFA (CONT.)

Calcodo della partizione degli stati di cui al Teorema di Myhill-Nerode:

$\Pi_{new} := \{Q - F, F\}$;

repeat

$\Pi := \Pi_{new}$; $\Pi_{new} := \varnothing$;

for S in Π do

partiziona S in $S_1, \ldots, S_m$

usando il più piccolo m t.c. $p, q \in S_i$ sse

$(\forall a \in \Sigma)(\exists S' \in \Pi)(\delta(p, a) \in S' \leftrightarrow \delta(q, a) \in S')$

$\Pi_{new} := \Pi_{new} \cup \{S_1, \ldots, S_m\}$

until $\Pi_{new} = \Pi$;

MINIMIZZAZIONE DI DFA (CONT.)

Definizione di M':

per S in Π sia $\bar{S} = \min_i\{q_i \in S\}$;

$Q' := \{\, \bar{S} \mid S \in \Pi \,\}$;

$\delta'(\bar{S}, a) := \min_i\{\, q_i \mid q_j \in S, \delta(q_j, a) = q_i \,\}$;

$F' := \{\, [S] \mid S \in \Pi, (\exists q \in S)(q \in F) \,\}$;

$q'_0 := q_0$;

si eliminino ricorsivamente da Q' e F'

gli stati diversi da q'_0 privi di archi entranti e

si eliminino da δ' gli archi uscenti da essi.

MINIMIZZAZIONE DI DFA (CONT.)

La minimizzazione dell'automa:

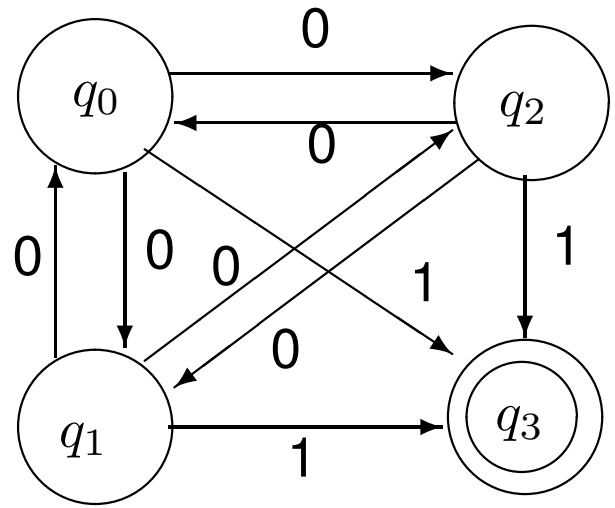
	0	1	
q_0	q_1	q_3	
q_1	q_2	q_3	
q_2	q_0	q_3	
q_3	q_3	q_3	F

conduce all'automa:

	0	1	
q_0	q_0	q_3	
q_3	q_3	q_3	F

MINIMIZZAZIONE DI DFA (CONT.)

Esercizio: Si determini, usando le tecniche di trasformazione viste finora, il DFA minimo equivalente all'automa:



Si determini inoltre, dimostrando in modo formale la propria affermazione, il linguaggio riconosciuto dal DFA calcolato.

MINIMIZZAZIONE DI DFA (CONT.)

Esercizio: Si determini l'automa minimo per il linguaggio denotato dall'espressione regolare:

$$(0^* + 1^* + (01)^*)$$

GRAMMATICHE LIBERE DAL CONTESTO

- ➜ Una grammatica è, intuitivamente, un insieme di regole che permettono di generare un linguaggio.
- ➜ Un ruolo fondamentale tra le grammatiche è costituito dalle grammatiche libere dal contesto.

Definizione 1. *Una grammatica libera dal contesto (CF) è una quadrupla* $G = \langle V, T, P, S \rangle$*, dove:*

- V *è un insieme finito di variabili, o simboli non terminali;*
- T *è un insieme finito di simboli terminali tale che* $V \cap T = \varnothing$*;*
- P *è un insieme finito di produzioni della forma* $A \to \alpha$ *dove:*
  - $A \in V$ *è una variabile, e*
  - $\alpha \in (V \cup T)^*$*;*
- $S \in V$ *è una variabile speciale, detta simbolo iniziale.*

GRAMMATICHE LIBERE DAL CONTESTO (ESEMPIO)

➜ Sia G la grammatica

$$G = \big\{\{E\}, \{\texttt{or}, \texttt{and}, \texttt{not}, (,), \texttt{0}, \texttt{1}\}, P, E\big\},$$

dove P è costituito dall'insieme di produzioni

$$E \to \texttt{0},$$
$$E \to \texttt{1},$$
$$E \to (E\ \texttt{or}\ E),$$
$$E \to (E\ \texttt{and}\ E),$$
$$E \to (\,\texttt{not}\ E).$$

➜ G genera delle espressioni booleane.

GRAMMATICHE LIBERE DAL CONTESTO (NOTAZIONE)

In generale:

- A, B, C, D, E, S denotano variabili;
- S denota il simbolo iniziale;
- $a, b, c, d, e, 0, 1$ denotano simboli terminali;
- X, Y, Z denotano simboli che possono essere sia terminali che variabili;
- u, v, w, x, y, z denotano stringhe di terminali;
- $\alpha, \beta, \gamma, \delta$ stringhe generiche di simboli (sia terminali che non);
- se $A \to \alpha_1, \ldots, A \to \alpha_n \in P$, esprimeremo questo fatto sinteticamente con $A \to \alpha_1 | \cdots | \alpha_n$;
- chiameremo A-produzione una generica produzione con la variabile A a sinistra.
- Esempio: $E \to \texttt{0} | \texttt{1} | (E \texttt{ or } E) | (E \texttt{ and } E) | (\texttt{ not } E)$.

GRAMMATICHE LIBERE DAL CONTESTO: DERIVAZIONE

Sia $G = \langle V, T, P, S \rangle$ una grammatica:

- Se $A \to \beta \in P$ e $\alpha, \gamma \in (V \cup T)^*$, allora $\alpha A \gamma \overset{G}{\Rightarrow} \alpha\beta\gamma$ ($\alpha A \gamma$ deriva immediatamente $\alpha\beta\gamma$).
- Se $\alpha_1, \ldots, \alpha_i \in (V \cup T)^*$, $i \geq 2$, e

$$\bigwedge_{j=1}^{i-1} \alpha_j \overset{G}{\Rightarrow} \alpha_{j+1},$$

  allora $\alpha_1 \overset{G}{\Rightarrow}_{i-1} \alpha_i$ (α_1 *deriva* α_i in $i-1$ passi).
- Per ogni $\alpha \in (V \cup T)^*$, $\alpha \overset{G}{\Rightarrow}_0 \alpha$.
- Se esiste i tale per cui $\alpha \overset{G}{\Rightarrow}_i \beta$, allora $\alpha \overset{G}{\Rightarrow}_* \beta$ (da α *deriva* β).

Nota:

- $\overset{G}{\Rightarrow}_*$ è la chiusura transitiva e riflessiva della relazione $\overset{G}{\Rightarrow}$;
- $\to$ denota l'appartenenza di una produzione all'insieme P;
- $\Rightarrow$ denota una relazione tra stringhe.

GRAMMATICHE LIBERE DAL CONTESTO: LINGUAGGIO GENERATO

- Sia $G = \langle V, T, P, S \rangle$ una grammatica. Il linguaggio generato da G è:

$$L(G) = \{ w \in T^* \mid S \overset{G}{\Rightarrow}_* w \}.$$

- L è un linguaggio libero dal contesto (CF) se esiste una grammatica CF G tale che $L = L(G)$.
- Due grammatiche G_1 e G_2 sono equivalenti se $L(G_1) = L(G_2)$.

GRAMMATICHE LIBERE DAL CONTESTO: ESEMPI

Esempio 1. *Σ^* è un linguaggio CF. Infatti, se $\Sigma = \{s_1, \ldots, s_n\}$, allora Σ^* è generato da $S \to \varepsilon | s_1 S | \cdots | s_n S$.*

Esempio 2. *La grammatica schematicamente definita da $S \to 0S1 \mid \varepsilon$ genera il linguaggio $\{\, 0^n 1^n \mid n \geq 0 \,\}$ (dimostrare per esercizio).*

Attenzione!

- ➜ Abbiamo dimostrato che quest'ultimo linguaggio non è un linguaggio regolare...
- ➜ ...pertanto l'insieme dei linguaggi regolari e quello dei linguaggi CF non sono lo stesso insieme.

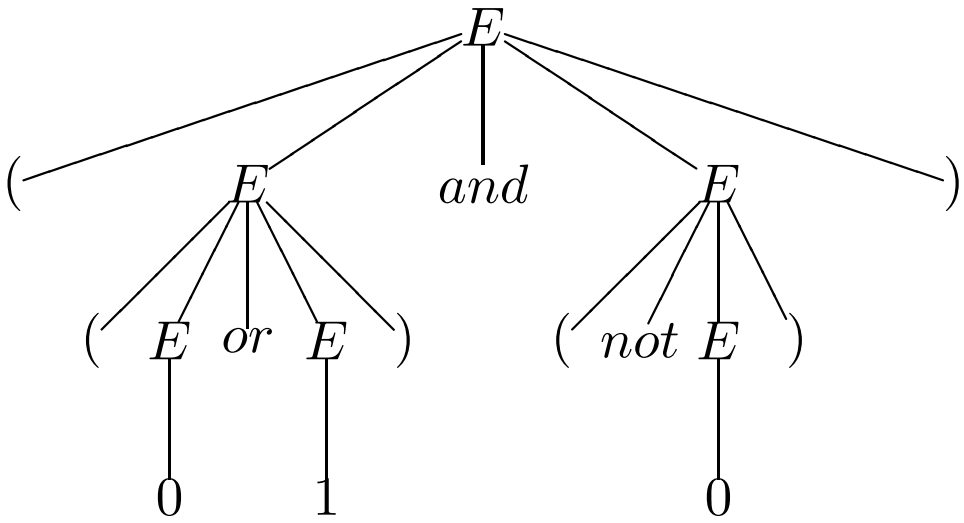
ALBERI DI DERIVAZIONE

Sia $G = \langle V, T, P, S \rangle$ una grammatica CF. Un albero è un albero di derivazione (parse tree) per G se:

1. ogni vertice ha una etichetta presa tra $V \cup T \cup \{\varepsilon\}$;
2. l'etichetta della radice appartiene a V;
3. ogni vertice interno (ovvero, non una foglia) ha etichetta appartenente a V;
4. se un vertice n è etichettato con A e $n_1, \ldots, n_k$ sono (ordinatamente, da sinistra a destra) i vertici figli di A etichettati con $X_1, \ldots, X_k$, allora $A \to X_1 \cdots X_k \in P$;
5. se un vertice n ha etichetta ε, allora n è una foglia ed è l'unico figlio di suo padre.

ALBERI DI DERIVAZIONE (CONT.)

- Gli alberi di derivazione sono rappresentazioni grafica delle derivazioni.
- Un albero che ha foglie etichettate con simboli non terminali, rappresenta una derivazione parziale.
- Un albero descrive una stringa $\alpha \in (V \cup T)^*$ se α è la stringa che si può leggere dalle etichette delle foglie da sinistra a destra.
- Esempio: un albero che descrive $((0\ \texttt{or}\ 1)\ \texttt{and}\ (\texttt{not}\ 0))$



DERIVAZIONE E ALBERI

Teorema. *Sia $G = \langle V, T, P, S\rangle$ una grammatica CF. Allora $S \overset{G}{\Rightarrow}_* \alpha$ se e solo se esiste un albero di derivazione con radice etichettata S per G che descrive α.*

Dimostrazione: Si dimostrerà un enunciato più forte:

> per ogni $A \in V$ si ha che $A \overset{G}{\Rightarrow}_* \alpha$ se e solo se esiste un albero di derivazione per G che descrive α la cui radice è etichettata A.

DERIVAZIONE E ALBERI ($\Longrightarrow$)

Per induzione sul numero i di passi di derivazione per $A \overset{G}{\Rightarrow}_i \alpha$.

Base: $i = 0$: $A \overset{G}{\Rightarrow}_0 \alpha$ implica, per definizione, che $\alpha = A$. Ma l'albero con unico nodo (radice) etichettato con A è un albero di derivazione (parziale) che descrive A stesso.

Passo: Supponiamo $A \overset{G}{\Rightarrow}_i \beta_1 B \beta_3 \overset{G}{\Rightarrow}_1 \underbrace{\beta_1\beta_2\beta_3}_{\alpha}$. Per ipotesi induttiva esiste un albero di derivazione T con radice etichettata A che descrive $\beta_1 B \beta_3$. Ma, per definizione, $\beta_1 B \beta_3 \overset{G}{\Rightarrow}_1 \beta_1\beta_2\beta_3$ se e solo se esiste la produzione $B \to \beta_2$ in P. Ma allora, applicando la regola (4) di definizione di albero, dall'albero T si ottiene l'albero che soddisfa i requisiti.

DERIVAZIONE E ALBERI ($\Longleftarrow$)

Per induzione sul numero di nodi interni dell'albero di derivazione etichettato in A che descrive α.

Completare per esercizio.

AMBIGUITÀ DELLE DERIVAZIONI

- Una derivazione è detta sinistra (leftmost) se ad ogni passo di una derivazione la produzione è applicata al simbolo non terminale più a sinistra.
- Una derivazione è detta destra (rightmost) se ad ogni passo di una derivazione la produzione è applicata al simbolo non terminale più a destra.
- Se $w \in L(G)$, dal teorema appena visto sappiamo che per essa esiste almeno un albero di derivazione con radice etichettata S.
- Fissato un albero di derivazione per w con radice etichettata S, si dimostra (farlo per esercizio) che esistono esattamente una derivazione sinistra ed esattamente una derivazione destra che gli corrispondono.
- In generale, un albero di derivazione rappresenta più derivazioni distinte di una stessa stringa.
- Il che non è un problema. . . il punto è che ci possono essere più alberi di derivazione per la stessa parola:

AMBIGUITÀ DELLE DERIVAZIONI (CONT.)

- Si consideri la grammatica

$$E \to E + E | E * E | 0 | 1 | 2 \,.$$

- Una derivazione sinistra è:

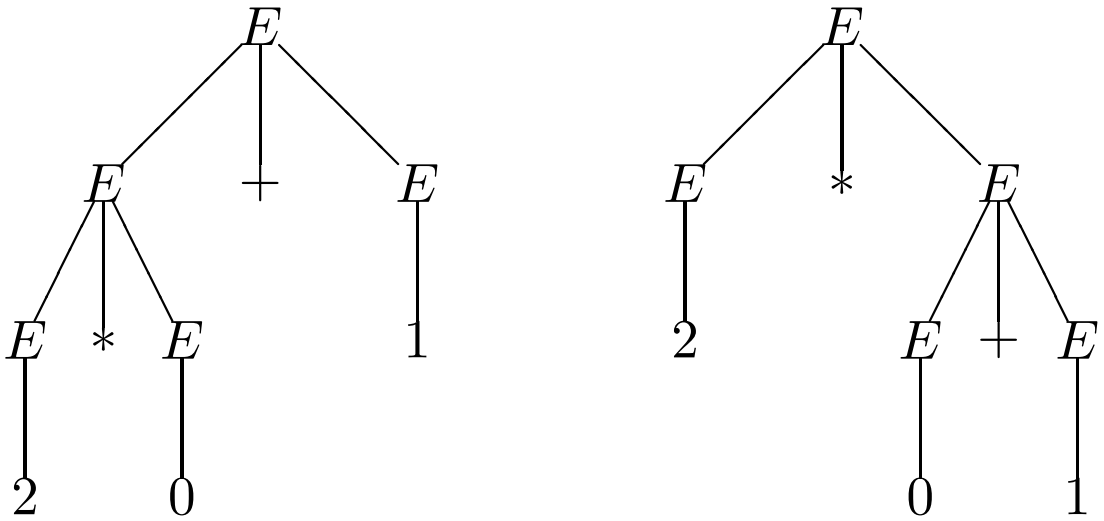
$$E \overset{G}{\Rightarrow} E + E \overset{G}{\Rightarrow} E * E + E \overset{G}{\Rightarrow}_3 2 * 0 + 1 \,.$$

- Un'altra derivazione sinistra per la stessa stringa è:

$$E \overset{G}{\Rightarrow} E * E \overset{G}{\Rightarrow} 2 * E \overset{G}{\Rightarrow} 2 * E + E \overset{G}{\Rightarrow}_2 2 * 0 + 1 \,.$$

- Gli alberi associati alle due derivazioni sono quindi diversi!

AMBIGUITÀ DELLE DERIVAZIONI (CONT.)



- L'albero a sinistra è intuitivamente associato a $(2 * 0) + 1 = 1$;
- quello a destra a $2 * (0 + 1) = 2$.

AMBIGUITÀ DEL LINGUAGGIO

- Una grammatica CF tale per cui esiste una parola con più di un albero di derivazione con radice etichettata S è detta ambigua.
- Un linguaggio CF per cui ogni grammatica che lo genera è ambigua è detto essere inerentemente ambiguo.
- Un esempio di linguaggio inerentemente ambiguo è:

$$\begin{aligned} L = &\{ a^n b^n c^m d^m : n \geq 1, m \geq 1\} \\ &\cup \{ a^n b^m c^m d^n : n \geq 1, m \geq 1\}. \end{aligned}$$

AMBIGUITÀ DEL LINGUAGGIO

➜ Un altro esempio di linguaggio inerentemente ambiguo è $L = L_1 \cup L_2$ con

$$L_1 = \{\, a^m b^m c^n \mid m, n \geq 0 \,\},$$
$$L_2 = \{\, a^m b^n c^n \mid m, n \geq 0 \,\}.$$

➜ È libero dal contesto, in quanto generato, tra le altre, dalla grammatica

$$S \to S_1 | S_2$$
$$S_1 \to A_1 C \qquad S_2 \to A B_1$$
$$A_1 \to \varepsilon | a A_1 b \qquad B_1 \to \varepsilon | b B_1 c$$
$$A \to \varepsilon | aA \qquad C \to \varepsilon | cC$$

➜ Informalmente: $L_1 \cap L_2 = \{\, a^m b^m c^m \mid m \geq 0 \,\}$ non è libero dal contesto, perciò nessuna grammatica libera da contesto *può decidere* come vanno interpretate le stringhe in $L_1 \cap L_2$.

SEMPLIFICAZIONE DI GRAMMATICHE CF

- È possibile vincolare la forma delle produzioni delle grammatiche CF senza alterare la classe dei linguaggi generabili.
- Mostreremo che ogni linguaggio CF può essere generato da una grammatica G dalle seguenti due proprietà:
  1. ogni variabile e ogni simbolo terminale di G compaiono in almeno una derivazione di una qualche parola di L;
  2. non ci sono produzioni della forma $A \to B$ con A e B variabili.
- Inoltre, se $\varepsilon \notin L$, si possono sempre evitare produzioni della forma $A \to \varepsilon$.
- Se $\varepsilon \notin L$, si può richiedere che ogni produzione sia della forma $A \to BC$ o $A \to a$ (forma normale di Chomsky).
- Se $\varepsilon \notin L$, si può richiedere che ogni produzione sia della forma $A \to a\alpha$ con $\alpha \in V^*$ (forma normale di Greibach).

ELIMINAZIONE DI SIMBOLI INUTILI

Sia $G = \langle V, T, P, S \rangle$ una grammatica CF.

- Un simbolo $X \in V \cup T$ è utile se esiste una derivazione

$$S \overset{G}{\Rightarrow}_* \alpha X \beta \overset{G}{\Rightarrow}_* w$$

  con $w \in T^*$.
- Altrimenti X è detto inutile.
- Mostreremo come i simboli inutili si possano eliminare in due passi:
  - prima si eliminano le variabili che non conducono in nessun modo ad una stringa di terminali;
  - poi si eliminano i simboli che non sono mai raggiunti da S.

ELIMINAZIONE DELLE VARIABILI IMPRODUTTIVE

Lemma 3. *Data una grammatica CF $G = \langle V, T, P, S\rangle$ tale che $L(G) \neq \varnothing$, si può calcolare in modo effettivo una grammatica equivalente $G' = \langle V', T, P', S\rangle$ (ovvero per cui $L(G') = L(G)$) tale che, per ogni $A \in V'$, esiste $w \in T^*$ tale che $A \overset{G'}{\Rightarrow}_* w$.*

Dimostrazione:

- Per calcolare V' usiamo il seguente operatore $\Gamma\colon \wp(V) \to \wp(V)$:

$$\Gamma(W) = \left\{ A \in V \;\middle|\; \exists \alpha \in (T \cup W)^* \,.\, (A \to \alpha) \in P \right\}$$

- Per induzione su $i \geq 1$, si mostra simultaneamente su tutti gli $A \in V$ che $A \in \Gamma^i(\varnothing)$ se e solo se esiste un albero di derivazione che descrive $w \in T^*$ con radice etichettata con A e di altezza $\leq i$ (completare per esercizio).
- Dunque, per il teorema sulla corrispondenza tra alberi e derivazioni, si ha che $A \in \Gamma^i(\varnothing)$ per qualche i sse $A \overset{G}{\Rightarrow}_* w$.

ELIMINAZIONE DELLE VARIABILI IMPRODUTTIVE (CONT.)

- Poiché V è finito e Γ monotono, si ha che esiste $i \leq |V|$ tale per cui

$$\Gamma^i(\varnothing) = \Gamma(\Gamma^i(\varnothing)) = \Gamma^{i+1}(\varnothing)(= \Gamma^{i+2}(\varnothing) = \cdots)$$

- Pertanto si riesce a calcolare finitamente l'insieme $V' = \Gamma^{|V|}(\varnothing)$, e $P' = \left\{ A \to \alpha \in P \;\middle|\; A \in V' \wedge \alpha \in (V' \cup T)^* \right\}$.
- L'ipotesi che il linguaggio sia non vuoto è indispensabile. Se infatti fosse $L = \varnothing$, il procedimento eliminerebbe il simbolo iniziale della grammatica, il che non è ammissibile.
- D'altra parte l'applicazione della procedura del lemma ad una grammatica permette di determinare se il linguaggio generato è vuoto o meno (lo è se e solo se la procedura eliminerebbe il simbolo iniziale).

ELIMINAZIONE DEI SIMBOLI IRRAGGIUNGIBILI DA S

Lemma 4. *Data una grammatica CF $G = \langle V, T, P, S\rangle$, si può calcolare in modo effettivo una grammatica equivalente $G' = \langle V', T', P', S\rangle$ tale che per ogni $X \in (V' \cup T')$ esistono α e β in $(V' \cup T')^*$ per cui $S \overset{G'}{\Rightarrow}_* \alpha X \beta$.*

Dimostrazione:

- Definiamo l'operatore $\Gamma\colon \wp(V \cup T) \to \wp(V \cup T)$ come

$$\Gamma(W) = \left\{ X \in V \cup T \;\middle|\; \exists A \in W\,.\,(A \to \alpha X \beta) \in P \right\} \cup \{S\}.$$

- Per induzione su $i \geq 0$, si mostra (farlo per esercizio) che $X \in \Gamma^i(\{S\})$ se e solo se esiste un albero di derivazione (parziale) con radice etichettata S e di altezza $\leq i$ in cui X compare come foglia.
- Per il teorema di corrispondenza tra alberi e derivazioni si ha che $X \in \Gamma^i(\{S\})$ per qualche $i \in \mathbb{N}$ se e solo se esistono α e β in $(V \cup T)^*$ per cui $S \overset{G}{\Rightarrow}_* \alpha X \beta$.

ELIMINAZIONE DI SIMBOLI IRRAGGIUNGIBILI DA S (CONT.)

- Γ è monotono, V e T sono finiti, dunque esiste $i \leq |V| + |T|$ tale che

$$\Gamma^i(\{S\}) = \Gamma^{i+1}(\{S\}) \left(= \Gamma^{i+2}(\{S\}) = \cdots\right)$$

- Siano dunque:
  - $V' = \Gamma^{|V|+|T|}(\{S\}) \cap V$;
  - $T' = \Gamma^{|V|+|T|}(\{S\}) \cap T$;
  - $P' = \left\{ (A \to \alpha) \in P \;\middle|\; A \in V' \wedge \alpha \in (V' \cup T')^* \right\}$.

Teorema. *Ogni linguaggio CF non vuoto è generato da una grammatica CF priva di simboli inutili.*

Dimostrazione: Immediato, applicando nell'ordine i due lemmi ora visti. (Perché?)

ELIMINAZIONE DI ε-PRODUZIONI

- Data una grammatica CF che genera un linguaggio L, si desidera eliminare le produzioni della forma $A \to \varepsilon$.
- Qualora $\varepsilon \in L$, ammetteremo la presenza della produzione $S \to \varepsilon$.
- Il metodo è quello di determinare quali variabili $A \in V$ sono tali che $A \overset{G}{\Rightarrow}_* \varepsilon$. Tali variabili sono dette annullabili.
- Il metodo consiste nell'eliminare, tra le X_i di ogni produzione $A \to X_1 \cdots X_n$ zero, una, ..., o tutte le variabili annullabili.
- Se togliendo tutte le variabili annullabili da $X_1 \cdots X_n$ la stringa diventasse vuota, allora anche A sarà annullabile (ma non aggiungeremo la produzione $A \to \varepsilon$, a meno che A non sia S).

ELIMINAZIONE DI ε-PRODUZIONI (CONT.)

Teorema. *Se $L = L(G)$ per qualche grammatica CF $G = \langle V, T, P, S\rangle$, allora $L \setminus \{\varepsilon\}$ è un linguaggio CF generato da una grammatica $G' = \langle V', T', P', S\rangle$ senza simboli inutili e senza ε-produzioni.*

Dimostrazione: Definiamo l'operatore $\Gamma\colon \wp(V) \to \wp(V)$ come

$$\Gamma(W) = \left\{ A \in V \;\middle|\; \exists A \to \alpha \in P \,.\, \alpha \setminus W = \varepsilon \right\}$$

dove, per ogni $W \in \wp(V)$,

$$\varepsilon \setminus W = \varepsilon,$$

$$X\alpha \setminus W = \begin{cases} \alpha \setminus W, & \text{se } X \in W; \\ X(\alpha \setminus W), & \text{se } X \notin W. \end{cases}$$

ELIMINAZIONE DI ε-PRODUZIONI (CONT.)

- Γ è monotono, V è finito, dunque esiste $i \leq |V|$ tale che $\Gamma^i(\varnothing) = \Gamma^{i+1}(\varnothing)$.
- Siano dunque $N = \Gamma^{|V|}(\varnothing)$ e

$$P'' = \left\{ A \to \alpha_1 \cdots \alpha_n \;\middle|\; \begin{array}{l} A \to X_1 \cdots X_n \in P, \\ X_i \notin N \implies \alpha_i = X_i, \\ X_i \in N \implies \alpha_i = X_i \vee \alpha_i = \varepsilon, \\ \alpha_1 \cdots \alpha_n \neq \varepsilon \end{array} \right\}$$

- E' facile mostrare (farlo per esercizio) che $L(G) \setminus \{\varepsilon\} = L\big(\langle V, T, P'', S\rangle\big)$.
- Applichiamo dunque il teorema sull'eliminazione dei simboli inutili alla grammatica $\langle V, T, P'', S\rangle$ per ottenere la grammatica desiderata $\langle V', T', P', S\rangle$.

ELIMINAZIONE DI ε-PRODUZIONI (CONT.)

- Ovviamente, se $\varepsilon \in L$, aggiungeremo la produzione $S \to \varepsilon$.
- Si osservi come la procedura individuata dalla dimostrazione permetta di capire se $\varepsilon \in L$.
- E' infatti sufficiente vedere se esiste in P una produzione $S \to A_1 \cdots A_k$ con le $A_i \in N$ per $i = 1, \ldots, k$, oppure direttamente la produzione $S \to \varepsilon$.

ELIMINAZIONE DI PRODUZIONI UNITARIE

➜ Una produzione della forma $A \to B$, con A e B variabili, si dice unitaria.

Teorema. *Ogni linguaggio CF L tale che $\varepsilon \notin L$ è generabile da una grammatica senza simboli inutili, senza ε-produzioni e senza produzioni unitarie.*

Dimostrazione:

➜ Grazie ai risultati già visti, possiamo partire da una grammatica senza simboli inutili e senza ε-produzioni.

➜ Innanzitutto si calcola, per ogni $A \in V$,

$$\text{seguenti}(A) = \{\, B \in V \mid A \overset{G}{\Rightarrow}_* B \,\}$$

➜ Poi si tolgono da P tutte le produzioni unitarie e, per ogni $A \in V$ e per ogni produzione non unitaria $B \to \beta \in P$ con $B \in \text{seguenti}(A)$ e $B \neq A$, si aggiunge la produzione $A \to \beta$.

ELIMINAZIONE DI PRODUZIONI UNITARIE (CONT.)

- L'equivalenza dei linguaggi si dimostra (farlo per esercizio) per induzione sul numero di produzioni unitarie presenti in una derivazione.
- Questa procedura potrebbe aver generato simboli inutili. Esempio: $P = \{S \to A, A \to a\}$.
  - Ma questi possono essere rimossi, come visto, e questo senza introdurre nuove produzioni unitarie.

FORMA NORMALE DI CHOMSKY

Teorema (Chomsky, 1959)**.** *Ogni linguaggio context-free L tale che $\varepsilon \notin L$ è generato da una grammatica in cui tutte le produzioni sono della forma $A \to BC$ e $A \to a$.*

Dimostrazione:

- ➜ Grazie ai risultati già visti, possiamo partire da una grammatica $G = \langle V, T, P, S \rangle$ senza simboli inutili, senza ε-produzioni e senza produzioni unitarie.
- ➜ Sia $A \to X_1 \cdots X_m \in P$ con $m \geq 1$ (non vi sono ε-produzioni):
- ➜ se $m = 1$, allora, poiché non vi sono produzioni unitarie, $X_1 \in T$: OK;
- ➜ se $m = 2$ e $X_1, X_2 \in V$: OK;

FORMA NORMALE DI CHOMSKY (CONT.)

- se $m \geq 2$ e qualcuno degli $X_i \in T$, allora per ogni $X_i \in T$ introduco in V una nuova variabile, diciamo B_i, aggiungo la produzione $B_i \to X_i$ e rimpiazzo la produzione $A \to X_1 \cdots X_m$ con $A \to Y_1 \cdots Y_m$, ove $Y_i = X_i$ se $X_i \in V$ e $Y_i = B_i$ altrimenti;
- se $m > 2$ e tutti gli X_i sono variabili, allora rimpiazzo la produzione $A \to X_1 \cdots X_m$ con le produzioni $A \to BX_3 \cdots X_m$ e $B \to X_1X_2$, dove B è una nuova variabile da aggiungere a V.
- È immediato verificare che il procedimento termina e che l'insieme finale di produzioni, P', è nella forma normale di Chomsky.

FORMA NORMALE DI CHOMSKY (CONT.)

- Resta da provare che $G' = \langle V', T, P', S \rangle$ (dove V' si ottiene da V con l'aggiunta delle nuove variabili) è equivalente a G. Ovvero che, per ogni $A \in V$,

$$A \stackrel{G}{\Rightarrow}_* w \qquad \Longleftrightarrow \qquad A \stackrel{G'}{\Rightarrow}_* w.$$

  - Si dimostra (farlo per esercizio) per induzione su $i \geq 1$ che $A \stackrel{G}{\Rightarrow}_i w$ implica che esiste $j \geq i$ tale che $A \stackrel{G'}{\Rightarrow}_j w$.
  - Viceversa, si dimostra (farlo per esercizio) per induzione su $i \geq 1$ che $A \stackrel{G'}{\Rightarrow}_i w$ implica che $A \stackrel{G}{\Rightarrow}_j w$ per qualche $j \leq i$.

FORMA NORMALE DI GREIBACH

Per raggiungere questa forma normale abbiamo bisogno di due lemmi preliminari.

Lemma 5 (Unfolding)**.** *Sia $G = \langle V, T, P, S\rangle$ una grammatica CF, sia $A \to \alpha B \gamma$ una produzione di P e $B \to \beta_1 | \cdots | \beta_n$ l'insieme delle B-produzioni. Sia $G' = \langle V, T, P', S\rangle$ ottenuta da G eliminando la produzione $A \to \alpha B \gamma$ da P e aggiungendo le produzioni $A \to \alpha\beta_1\gamma | \cdots | \alpha\beta_n\gamma$. Allora $L(G) = L(G')$.*

Dimostrazione: Per esercizio.

FORMA NORMALE DI GREIBACH (CONT.)

Lemma 6 (Eliminazione ricorsione sinistra)**.** *Sia $G = \langle V, T, P, S\rangle$ una grammatica CF e sia $A \to A\alpha_1 | \cdots | A\alpha_m$ l'insieme delle A-produzioni di G per cui A è anche il simbolo più a sinistra della stringa di destra delle produzioni. Siano $A \to \beta_1 | \cdots | \beta_n$ le rimanenti A-produzioni. G è equivalente a $G' = \langle V \cup \{B\}, T, P', S\rangle$ dove si è aggiunta una nuova variabile B a V e si sono rimpiazzate tutte le A-produzioni con*

1. $A \to \beta_i | \beta_i B$ *per* $i \in \{1, \ldots, n\}$*;*
2. $B \to \alpha_i | \alpha_i B$ *per* $i \in \{1, \ldots, m\}$*.*

Dimostrazione:

- Mostriamo prima che se $x \in L(G)$ allora $x \in L(G')$.
- Sia T_S un albero di derivazione per x in G.

FORMA NORMALE DI GREIBACH (CONT.)

- Sia T_A un sottoalbero di T_S radicato in un nodo v etichettato con A tale che v non è il figlio sinistro di un nodo etichettato con A.
- Il sottoalbero di T_A che comprende tutte e sole le applicazioni di A-produzioni a nodi in posizione leftmost ha una frontiera della forma $\beta_j \alpha_{i_1} \cdots \alpha_{i_k}$, con $k \geq 0$, $1 \leq j \leq n$ e $\{i_1, \ldots, i_k\} \subseteq \{1, \ldots, m\}$.
- L'albero T'_A, radicato in v' etichettato con A e ottenuto dalla produzione $A \to \beta_j$ (se $k = 0$) oppure da $A \to \beta_j B$ e poi espandendo il B-nodo rightmost con $B \to \alpha_{i_1}, \ldots, B \to \alpha_{i_k}$ è un albero di derivazione per G' con la stessa frontiera di T_A.
- Siccome la sostituzione di T_A con T'_A si può effettuare per ogni scelta di T_A in T_S, iterando il procedimento si ottiene un albero T'_S che è di derivazione per G' e la cui frontiera è x.
- Completare e dimostrare l'implicazione inversa ($x \in L(G') \implies x \in L(G)$) per esercizio.

FORMA NORMALE DI GREIBACH (CONT.)

Teorema (Greibach, 1965)**.** *Ogni linguaggio context-free L tale che $\varepsilon \notin L$ è generato da una grammatica in cui tutte le produzioni sono della forma $A \to a\alpha$, ove α è una stringa (eventualmente vuota) di simboli non terminali.*

Dimostrazione: Sia G una grammatica in forma normale di Chomsky. Sia $V = \{A_1, \ldots, A_m\}$. Si costruirà in tre passi una grammatica G' in forma normale di Greibach equivalente a G.

FORMA NORMALE DI GREIBACH (PASSO 1)

➜ Si applica il seguente algoritmo:

for $k := 1$ to m do
begin
  for $j := 1$ to $k - 1$ do
    elimina le produzioni $A_k \to A_j\alpha$ mediante unfolding;
  elimina le produzioni $A_k \to A_k\alpha$ mediante elim. ricors. sinistra
end;

➜ A causa dell'eliminazione della ricorsione sinistra sono state introdotte delle nuove variabili: $B_1, \ldots, B_h$.

➜ L'equivalenza tra G e la grammatica ottenuta deriva dai lemmi appena dimostrati.

FORMA NORMALE DI GREIBACH (PASSO 1, CONT.)

➜ A questo punto tutte le produzioni sono della forma:

$$\begin{aligned} A_i &\rightarrow A_j\alpha \quad j > i \\ A_i &\rightarrow a\beta \quad a \in T \\ B_i &\rightarrow \gamma \end{aligned}$$

➜ Poiché siamo partiti da una grammatica in forma normale di Chomsky, si osserva che α è una stringa di variabili.

➜ Per lo stesso motivo, β e γ sono stringhe di variabili.

➜ Per quanto riguarda le stringhe γ, esse non iniziano mai con un B_j.

FORMA NORMALE DI GREIBACH (PASSO 2)

- Ogni produzione $A_m \to \alpha$ ha la parte destra iniziante con un simbolo terminale.
- Posso dunque effettuare l'unfolding delle A_i-produzioni rimpiazzando A_m con ciascuna delle sue possibili parti destre.
- Posso iterare questo procedimento all'indietro, ottenendo produzioni della forma:

$$A_i \quad \to \quad a\beta \quad a \in T$$

per ogni $i = 1, \ldots, m$, con β stringa di variabili.

FORMA NORMALE DI GREIBACH (PASSO 3)

➜ Si tratta ora di semplificare le produzioni $B_i \to \gamma$. Applicando la sostituzione sul primo (eventuale) simbolo A_i della stringa, si giunge al risultato.

FORMA NORMALE DI GREIBACH (ESEMPIO)

- Sia $G = \big(\{A_1, A_2, A_3\}, \{0, 1\}, P, A_1\big)$, con
  $P = \{A_1 \to A_2A_3, A_2 \to A_3A_1 \mid 1, A_3 \to A_1A_2 \mid 0.$
- Solo A_3 viola la condizione sull'ordinamento, dunque solo le produzioni per A_3 devono essere modificate, e diventano

$$A_3 \to A_3A_1A_3A_2 \mid 1A_3A_2 \mid 0.$$

- L'eliminazione della ricorsione sinistra ci porta a

$$A_3 \to 1A_3A_2B_1 \mid 0B_1 \mid 1A_3A_2 \mid 0,$$
$$B_1 \to A_1A_3A_2 \mid A_1A_3A_2B_1.$$

  Tutte le A_3-produzioni iniziano con un simbolo terminale.
- Facciamo l'unfolding di A_3 sull'unica A_2-produzione che abbiamo, ed ora anche questa inizia con un terminale.
- Facciamo l'unfolding di A_2 sull'unica A_1-produzione che abbiamo, ed ora anche questa inizia con un terminale.

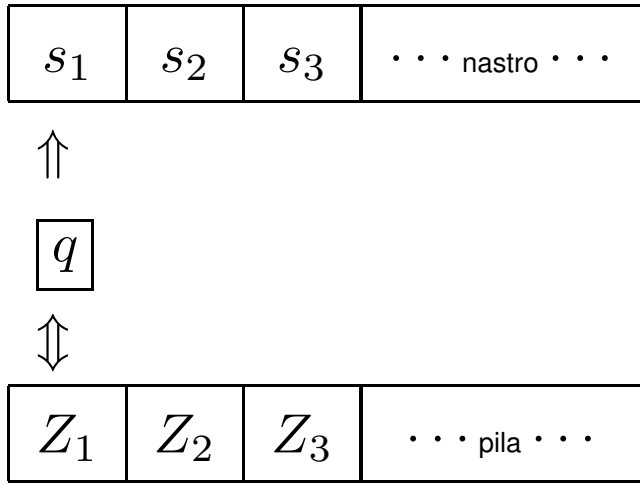
FORMA NORMALE DI GREIBACH (CONT.)

- Un importante caso particolare di grammatiche in forma normale di Greibach sono le grammatiche CF in cui le produzioni hanno tutte la forma $A \to a$ oppure $A \to aB$.
- Tali grammatiche sono dette lineari destre ed hanno l'importante caratteristica di generare esattamente i linguaggi regolari.

AUTOMI A PILA

- La pila (in inglese stack) è una struttura dati LIFO (*Last In First Out*).
- Un automa a pila è una macchina costituita da:
  1. un controllo che è un automa a stati finiti;
  2. un nastro di input che, come negli NFA e DFA, è di sola lettura;
  3. una pila sulla quale sono possibili le seguenti operazioni:
     - $\mathrm{push}(\alpha)$: *spinge* una stringa α in testa alla pila;
     - $X = \mathrm{pop}()$: *preleva* il simbolo in testa alla pila;
     - $\mathrm{empty}()$: vero se e solo se se la pila è vuota.

AUTOMI A PILA (CONT.)



AUTOMI A PILA (CONT.)

Le azioni possibili di un automa a pila sono:

- ➜ Leggere dal nastro e dalla pila (con l'operazione pop) contemporaneamente:
  - avanzare la testina a destra sul nastro;
  - cambiare stato nel controllo;
  - inserire una stringa (anche vuota) sulla pila (con l'operazione push).
- ➜ Leggere solo dalla pila:
  - cambiare stato nel controllo,
  - inserire una stringa (anche vuota) sulla pila.

AUTOMI A PILA: DEFINIZIONE FORMALE

Definizione 2. *Un automa a pila non-deterministico (APND) è una 7-upla:* $M = \langle Q, \Sigma, R, \delta, q_0, Z_0, F \rangle$ *dove:*

- Q *è un insieme finito di stati;*
- Σ *è l'alfabeto (finito) di input;*
- R *è l'alfabeto (finito) della pila;*
- $q_0 \in Q$ *è lo stato iniziale;*
- $Z_0 \in R$ *è il simbolo iniziale sulla pila;*
- $F \subseteq Q$ *è l'insieme degli stati finali;*
- $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \times R \to \wp_f(Q \times R^*)$ *è la funzione di transizione.*

AUTOMI A PILA: DESCRIZIONE ISTANTANEA

Definizione 3. *Una descrizione istantanea per un APND è una tripla*

$$(q, x, \gamma)$$

dove

- $q \in Q$ *è lo stato corrente;*
- $x \in \Sigma^*$ *è la stringa ancora da leggere sul nastro;*
- $\gamma \in R^*$ *è la sequenza di simboli contenuti sulla pila.*

Se $(p, \gamma) \in \delta(q, a, Z)$*, un passo di computazione dell'APND è definito da*

$$(q, aw, Z\alpha) \mapsto_M (p, w, \gamma\alpha), \quad \text{con } a \in \Sigma \cup \{\varepsilon\}$$

AUTOMI A PILA: LINGUAGGIO ACCETTATO

Il linguaggio $L(M)$ riconosciuto da un APND M si può definire in due modi diversi:

1. per pila vuota:

$$L_p(M) = \left\{ x \in \Sigma^* \;\middle|\; \exists q \in Q \,.\, (q_0, x, Z_0) \mapsto_M^* (q, \varepsilon, \varepsilon) \right\};$$

2. o per stato finale:

$$L_F(M) = \left\{ x \in \Sigma^* \;\middle|\; \exists q \in F \,.\, \exists \gamma \in R^* \,.\, (q_0, x, Z_0) \mapsto_M^* (q, \varepsilon, \gamma) \right\}$$

Proposizione 1. *Per ogni APND M, esiste un APND M' tale che $L_F(M) = L_p(M')$.*

Quando l'APND è previsto per riconoscere le stringhe per pila vuota non è restrittivo assumere $F = \varnothing$.

AUTOMI A PILA: ESEMPIO

L'APND

$$M = \langle \{q_0, q_1\}, \{a, b, c\}, \{Z, A, B\}, \delta, q_0, Z, \varnothing \rangle$$

dove

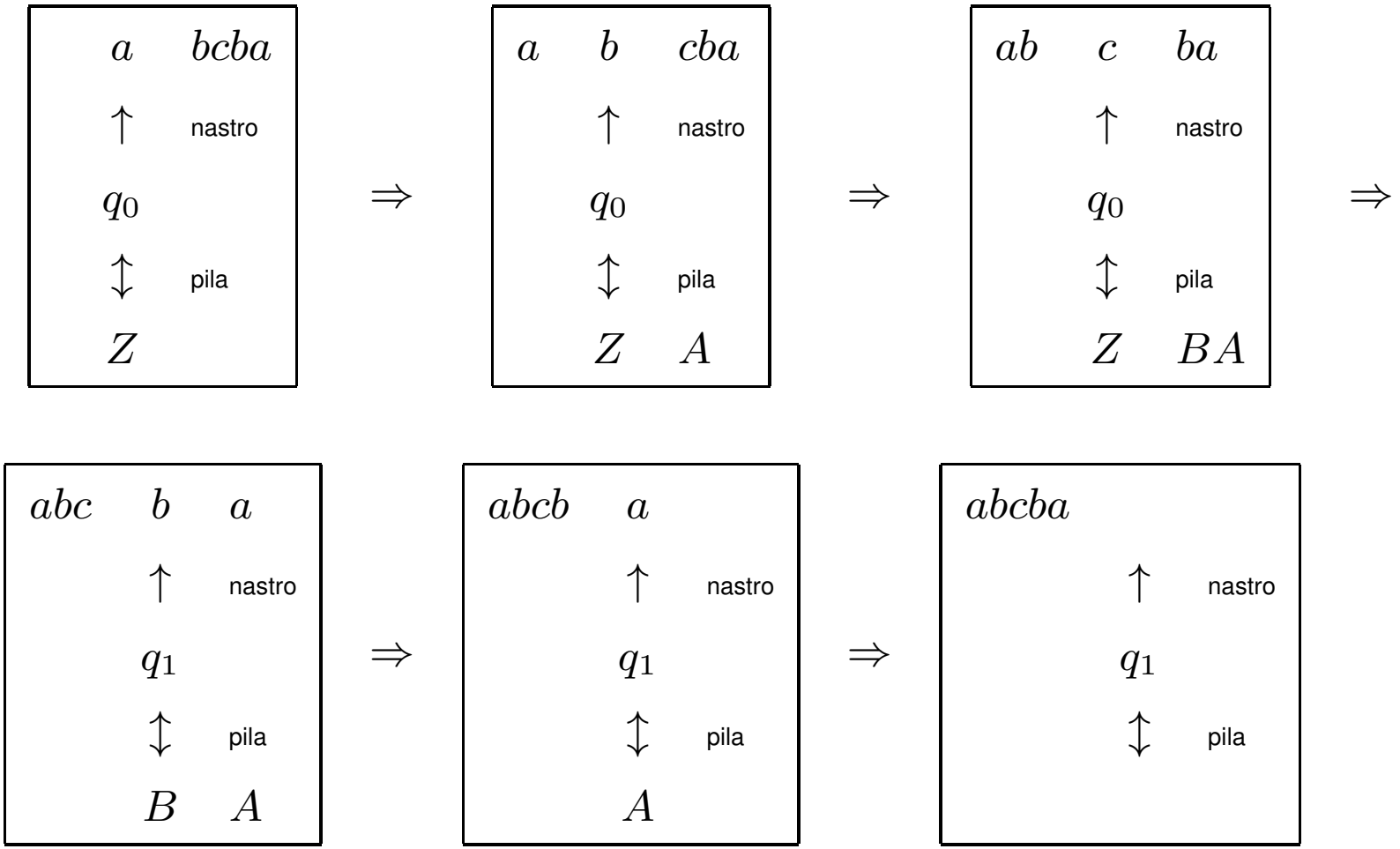
q_0	ε	a	b	c
Z		q_0, ZA	q_0, ZB	q_1, ε
A				
B				

q_1	ε	a	b	c
Z		q_1, Z	q_1, Z	
A		q_1, ε	q_1, Z	q_1, Z
B		q_1, Z	q_1, ε	q_1, Z

riconosce per pila vuota il linguaggio xcx^{r}, dove $x \in \{a, b\}^*$, $c \notin \{a, b\}$ e x^{r} è la stringa x rovesciata (ovvero letta da destra verso sinistra):

$$\varepsilon^{\mathrm{r}} = \varepsilon;$$
$$(aw)^{\mathrm{r}} = (w^{\mathrm{r}})a.$$

AUTOMI A PILA: ESEMPIO (RICONOSCIMENTO DI $abcba$)



AUTOMI A PILA: ESEMPIO (RICONOSCIMENTO DI $abcba$)

- Al contrario di quanto accade per gli automi a stati finiti gli automi a pila non-deterministici sono strettamente più potenti di quelli deterministici.
- Nel caso del linguaggio dell'esempio precedente, un automa a pila deterministico è sufficiente per via del *separatore* c.
- Grazie ed esso le fasi di caricamento della pila (la lettura della stringa x) ed il suo scaricamento (la lettura della stringa x^{r}) sono univocamente delimitate.
- Gli automi a pila deterministici sono solitamente utilizzati per il riconoscimento sintattico dei linguaggi di programmazione.
- Si osservi come, anche per questo motivo, la sintassi degli usuali linguaggi di programmazione sia strutturata in modo tale da intercalare parole chiave come `if`, `while`, `then`, `...`

AUTOMI A PILA: ALTRO ESEMPIO

L'APND

$$M = \langle \{q_0, q_1\}, \{a, b\}, \{Z, A, B\}, \delta, q_0, Z, \varnothing \rangle$$

dove

q_0	ε	a	b
Z	q_1, ε	q_0, AZ	q_0, BZ
A		q_0, AA	q_0, BA
		q_1, ε	
B		q_0, AB	q_0, BB
			q_1, ε

q_1	ε	a	b
Z	q_1, ε		
A		q_1, ε	
B			q_1, ε

riconosce il linguaggio delle stringhe palindrome sull'alfabeto $\{a, b\}$:
$L = \left\{ ww^{\mathrm{r}} \;\middle|\; w \in \{a, b\}^* \right\}$.

AUTOMI A PILA: RIDUZIONE DEL NUMERO DEGLI STATI

Teorema. *Se $L = L_p(M)$ con M APND, allora $L = L_p(M')$ con M' APND con 1 solo stato.*

Dim.: Siano $M = \langle Q, \Sigma, R, \delta, q_0, Z_0, \varnothing \rangle$, $R' = \{Z_0\} \cup (Q \times R \times Q)$, e

$$M' = \langle \{\blacklozenge\}, \Sigma, R', \delta', \blacklozenge, Z_0, \varnothing \rangle.$$

Per $c \in \Sigma \cup \{\varepsilon\}$ e per ogni transizione di M

$$(p_0, B_1 \dots B_m) \in \delta(p, c, A)$$

introduciamo $|Q|^m$ transizioni in M': per ogni $p_1, \dots, p_m \in Q$,

$$\big(\blacklozenge, (p_0, B_1, p_1)(p_1, B_2, p_2) \dots (p_{m-1}, B_m, p_m)\big) \in \delta'\big(\blacklozenge, c, (p, A, p_m)\big);$$

se $p = q_0$ e $A = Z_0$ introduciamo inoltre le $|Q|^m$ transizioni

$$\big(\blacklozenge, (p_0, B_1, p_1)(p_1, B_2, p_2) \dots (p_{m-1}, B_m, p_m)\big) \in \delta'(\blacklozenge, c, Z_0).$$

AUTOMI A PILA: RIDUZIONE DEL NUMERO DEGLI STATI (CONT.)

M' simula M "indovinando nondeterministicamente" gli stati futuri di M e "filtrando" poi solo le scelte corrette.

Dimostreremo che, per ogni $w, x \in \Sigma^*$, ogni $m \in \mathbb{N}$, ogni $B_1, \ldots, B_m \in R$,

$$(q_o, w, Z_0) \mapsto_M^+ (p_0, x, B_1 \ldots B_m)$$

se e solo se esistono $p_1, \ldots, p_m \in Q$ tali che

$$(\blacklozenge, w, Z_0) \mapsto_{M'}^+ \big(\blacklozenge, x, (p_0, B_1, p_1) \ldots (p_{m-1}, B_m, p_m)\big).$$

Questo implica, in particolare, che, per ogni $w \in \Sigma^*$,

$$(q_o, w, Z_0) \mapsto_M^+ (p_0, \varepsilon, \varepsilon) \quad \Longleftrightarrow \quad (\blacklozenge, w, Z_0) \mapsto_{M'}^+ (\blacklozenge, \varepsilon, \varepsilon)$$

ovvero $L(M) = L(M')$.

AUTOMI A PILA: RIDUZIONE DEL NUMERO DEGLI STATI (CONT.)

Dimostriamo, per induzione sulla lunghezza n delle derivazioni, che, per ogni $w, x \in \Sigma^*$,

$$(q_o, w, Z_0) \mapsto_M^n (p_0, x, B_1 \dots B_m)$$

implica che esistono $p_1, \dots, p_m \in Q$ tali che

$$(\blacklozenge, w, Z_0) \mapsto_{M'}^n \big(\blacklozenge, x, (p_0, B_1, p_1) \dots (p_{m-1}, B_m, p_m)\big).$$

L'implicazione inversa è lasciata per esercizio.

AUTOMI A PILA: RIDUZIONE DEL NUMERO DEGLI STATI (CONT.)

Caso base, $n = 1$: per $c \in \Sigma \cup \{\varepsilon\}$,

$$(q_o, cx, Z_0) \mapsto_M (p_0, x, B_1 \dots B_m)$$

implica

$$(p_0, B_1 \dots B_m) \in \delta(q_o, c, Z_0),$$

il che implica che esistono $p_1, \dots, p_m \in Q$ tali che

$$\big(\blacklozenge, (p_0, B_1, p_1) \dots (p_{m-1}, B_m, p_m)\big) \in \delta'(\blacklozenge, c, Z_0)$$

e dunque

$$(\blacklozenge, cx, Z_0) \mapsto_{M'} \big(\blacklozenge, x, (p_0, B_1, p_1) \dots (p_{m-1}, B_m, p_m)\big).$$

AUTOMI A PILA: RIDUZIONE DEL NUMERO DEGLI STATI (CONT.)

Passo induttivo, $n = t + 1$: per $c \in \Sigma \cup \{\varepsilon\}$,

$$(q_o, w, Z_0) \mapsto_M^t (p'_0, cx, A_1 \dots A_r) \mapsto_M (p''_0, x, B_1 \dots B_m A_2 \dots A_r)$$

implica:

① esistono $p'_1, \dots, p'_r \in Q$ tali che

$$(\blacklozenge, w, Z_0) \mapsto_{M'}^t \big(\blacklozenge, cx, (p'_0, A_1, p'_1) \dots (p'_{r-1}, A_r, p'_r)\big).$$

② $(p''_0, B_1 \dots B_m) \in \delta(p'_0, c, A_1)$, ovvero esistono $p''_1, \dots, p''_m \in Q$, con $p''_m = p'_1$, tali che

$$\big(\blacklozenge, (p''_0, B_1, p''_1) \dots (p''_{m-1}, B_m, p''_m)\big) \in \delta'\big(\blacklozenge, c, (p'_0, A_1, p'_1)\big)$$

e dunque

$$\big(\blacklozenge, cx, (p'_0, A_1, p'_1) \ldots (p'_{r-1}, A_r, p'_r)\big)$$
$$\mapsto_{M'} \big(\blacklozenge, x, (p''_0, B_1, p''_1) \ldots (p''_{m-1}, B_m, p''_m)(p'_1, A_2, p'_2) \ldots (p'_{r-1}, A_r, p'_r)\big).$$

Teorema. *Se $\varepsilon \notin L = L_p(M)$ con M APND con 1 stato, L è CF.*

Dimostrazione:

- Sia $M = \langle \{q\}, \Sigma, R, \delta, q, Z_0, \varnothing \rangle$ un APND con 1 solo stato che riconosce il linguaggio $L_p(M)$.
- Definiamo la grammatica libera da contesto $G = \langle R, \Sigma, Z_0, P \rangle$, dove R sono i simboli non terminali, $Z_0 \in R$ è il simbolo iniziale e

$$P = \{\, Z \to aZ_1Z_2\cdots Z_n \;\big|\; (q, Z_1Z_2\cdots Z_n) \in \delta(q, a, Z) \,\}.$$

- È ovvio che se $\alpha \in R^*$, allora:

$$(q, a, Z\alpha) \mapsto_M (q, \varepsilon, Z_1Z_2\cdots Z_n\alpha) \iff Z\alpha \overset{G}{\Rightarrow} aZ_1Z_2\cdots Z_n\alpha$$

- Per induzione si dimostra che per ogni $x \in \Sigma^*$, $x \neq \varepsilon$:

$$(q, x, Z_0) \mapsto_M^* (q, \varepsilon, \varepsilon) \iff Z_0 \overset{G}{\Rightarrow}_* x.$$

AUTOMI A PILA E LINGUAGGI CF

Esercizio: Generalizzare la dimostrazione precedente per trattare anche il caso in cui $\varepsilon \in L$.

AUTOMI A PILA E LINGUAGGI CF

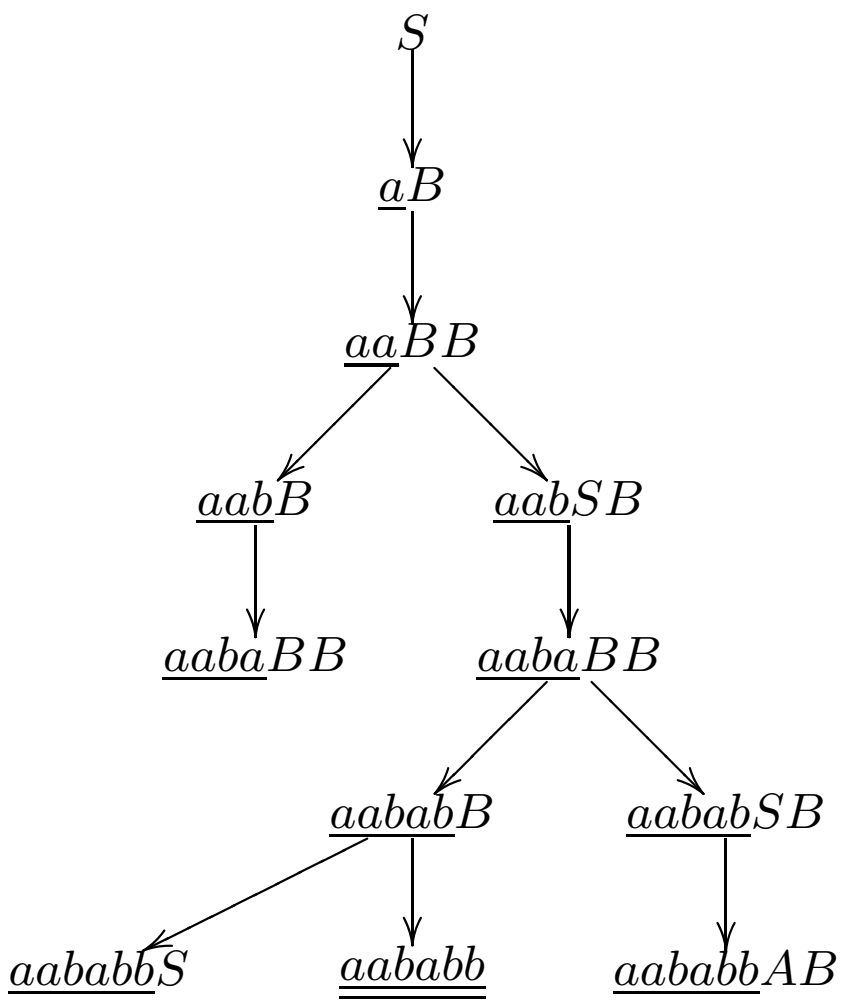
- C'è una forte analogia tra la derivazione da una grammatica CF ed i passi di caricamento/svuotamento della pila durante le varie fasi di riconoscimento delle sottostringhe di una data stringa da parte di un APND.
- Si consideri la seguente grammatica G in forma normale di Greibach:

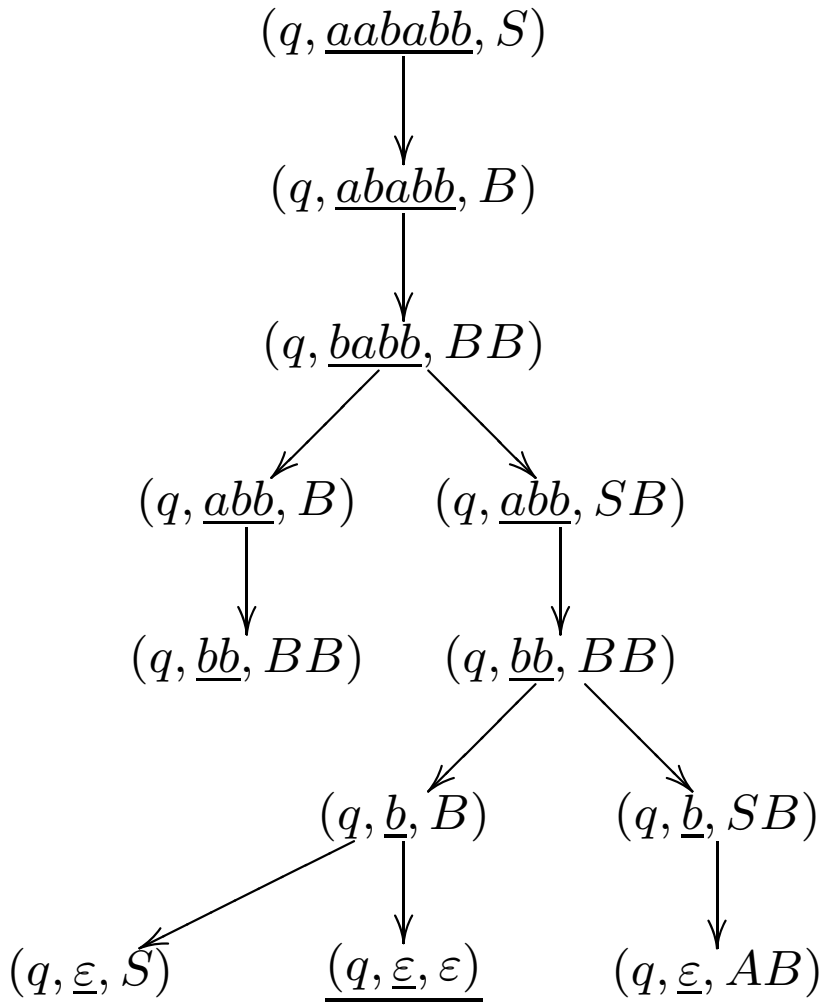
$$
\begin{array}{lcl@{\qquad}lcl}
S & \to & aB & A & \to & a \\
S & \to & bA & B & \to & bS \\
A & \to & aS & B & \to & aBB \\
A & \to & bAA & B & \to & b
\end{array}
$$

AUTOMI A PILA E LINGUAGGI CF (CONT.)

➜ Sia M l'APND con 1 solo stato q definito dalla seguente relazione di transizione:

q	ε	a	b
S		q, B	q, A
A		q, ε q, S	q, AA
B		q, BB	q, ε q, S


S
$\underline{a}B$
$\underline{aa}BB$
$\underline{aab}B$
$\underline{aab}SB$
$\underline{aaba}BB$
$\underline{aaba}BB$
$\underline{aabab}B$
$\underline{aabab}SB$
$\underline{aababb}S$
$\underline{aababb}$
$\underline{aababb}AB$


$(q, \underline{aababb}, S)$
$(q, \underline{ababb}, B)$
$(q, \underline{babb}, BB)$
$(q, \underline{abb}, B)$
$(q, \underline{abb}, SB)$
$(q, \underline{bb}, BB)$
$(q, \underline{bb}, BB)$
$(q, \underline{b}, B)$
$(q, \underline{b}, SB)$
$(q, \underline{\varepsilon}, S)$
$(q, \underline{\varepsilon}, \varepsilon)$
$(q, \underline{\varepsilon}, AB)$

AUTOMI A PILA E LINGUAGGI CF (CONT.)

Teorema. *Se $L \not\ni \varepsilon$ è CF, allora esiste un APND M tale che $L = L_p(M)$.*

Dimostrazione:

- Sia $L = L(G)$ con $G = \langle V, T, P, S\rangle$ grammatica CF in forma normale di Greibach.
- Definiamo: $Q = \{q\}$, $\Sigma = T$, $R = V$, $Z_0 = S$, $F = \varnothing$, e

$$(q, \alpha) \in \delta(q, a, A) \iff A \to a\alpha \in P.$$

- Si ha una derivazione in G del tipo

$$xA\beta \overset{G}{\Rightarrow} xa\alpha\beta, \qquad \text{con } \alpha, \beta \in V^* \text{ e } x \in T^*,$$

se e solo se una mossa di M tale che

$$(q, a, A\beta) \mapsto_M (q, \varepsilon, \alpha\beta)$$

AUTOMI A PILA E LINGUAGGI CF (CONT.)

➜ Per induzione sul sul numero di passi della derivazione (completare per esercizio) si ha che

$$xA\beta \overset{G}{\Rightarrow}_* xy\alpha\beta, \qquad \text{con } A \in V,\ \alpha, \beta \in V^*\ \text{e}\ x, y \in T^*,$$

se e solo se

$$(q, y, A\beta) \mapsto^*_M (q, \varepsilon, \alpha\beta).$$

➜ Ne consegue che: $S \overset{G}{\Rightarrow}_* y$ se e solo se $(q, y, S) \mapsto^*_M (q, \varepsilon, \varepsilon)$.

AUTOMI A PILA E LINGUAGGI CF (CONT.)

Esercizio: Generalizzare la dimostrazione precedente per trattare anche il caso in cui $\varepsilon \in L$.

IL PUMPING LEMMA PER I LINGUAGGI CF

Lemma 7 (Bar-Hillel, Perles, Shamir, 1961)**.** *Sia L un linguaggio CF. Allora esiste una costante $n \in \mathbb{N}$ (dipendente da L) tale che per ogni $z \in L$ con $|z| \geq n$, esistono stringhe u, v, w, x e y tali che*

1. $z = uvwxy$,
2. $|vx| \geq 1$,
3. $|vwx| \leq n$, *e*
4. *per ogni $i \geq 0$ vale che $uv^iwx^iy \in L$.*

Dimostrazione:

- ➜ Dimostriamo che il teorema vale se $\varepsilon \notin L$ (la dimostrazione è appena più complicata se $\varepsilon \in L$).
- ➜ Sia $G = \langle V, T, P, S \rangle$ una grammatica in forma normale di Chomsky che genera L.

IL PUMPING LEMMA PER I LINGUAGGI CF (CONT.)

- Per induzione su $i \geq 1$, si dimostra (farlo per esercizio) che se un albero di derivazione di G per una stringa $z \in T^*$ ha tutti i cammini di lunghezza minore o uguale a i, allora $|z| \leq 2^{i-1}$.
- Sia $|V| = k$ e sia $n = 2^k$ (si noti che $k \geq 1$ e $n \geq 2$).
- Se $z \in L$ e $|z| \geq n$, allora ogni albero di derivazione per z deve avere un cammino di lunghezza almeno $k + 1$.
- Ma tale cammino ha almeno $k + 2$ nodi, tutti, eccetto l'ultimo, etichettati da variabili. Pertanto vi deve essere almeno una variabile ripetuta nel cammino.
- In particolare, dunque, vi devono essere due nodi v_1 e v_2 tali che:
  1. entrambi sono etichettati dalla stessa variabile, diciamo A;
  2. v_1 è più vicino alla radice di v_2;
  3. poiché $n \geq 2$, a v_1 è associata una produzione del tipo: $A \to BC$;
  4. il cammino da v_1 a qualsiasi foglia è lungo al più $k + 1$.

IL PUMPING LEMMA PER I LINGUAGGI CF (CONT.)

- Si prenda ora il sottoalbero T_1 con radice in v_1: esso rappresenta una derivazione di una parola z_1 di lunghezza al più 2^k.
- Sia z_2 invece la parola relativa al sottoalbero T_2 di T_1 con radice in v_2.
- Possiamo scrivere $z_1 = vz_2x$.
- Inoltre almeno una tra v e x deve essere non vuota, in quanto a v_1 è associata una produzione del tipo: $A \to BC$ ed il sottoalbero T_2 è un sottoalbero di esattamente uno dei due sottoalberi individuati da B e C.
- Dunque

$$A \overset{G}{\Rightarrow}_* vAx \qquad \text{e} \qquad A \overset{G}{\Rightarrow}_* z_2$$

  dove $|vz_2x| \leq 2^k = n$.
- Ma allora abbiamo anche che $A \overset{G}{\Rightarrow}_* v^i z_2 x^i$ per ogni $i \geq 0$.
- Si ponga $w = z_2$ e u e y il prefisso e il suffisso di z necessari affinchè $z = uvwxy$.

IL PUMPING LEMMA PER I LINGUAGGI CF (ESEMPIO)

- Il linguaggio $\{ a^i b^i c^i \mid i \geq 1 \}$ non è CF.
- Per ogni $n \geq 1$ scegliamo $z = a^n b^n c^n$, e dunque $z \in L$ e $|z| \geq n$,
- Dobbiamo mostrare che, per ogni quintupla di stringhe $u, v, w, x, y \in T^*$ tali che $a^n b^n c^n = uvwxy$, $|vx| \geq 1$ e $|vwx| \leq n$, esiste un $i \geq 0$ tale per cui $uv^i wx^i y \notin L$.
- Vi sono molti modi di partizionare z in $uvwxy$ che soddisfano i requisiti. Tuttavia questi si possono raggruppare nelle seguenti casistiche:
  1. $uvwx = a^h$ con $h \leq n$
  2. $u = a^h$, $vwx = a^k b^m$ con $k > 0$, $k + m \leq n$
  3. $u = a^n b^h$, $vwx = b^m$ con $m \leq n$
  4. $u = a^n b^h$, $vwx = b^k c^m$ con $k + m \leq n$
  5. $u = a^n b^n c^h$, $vwx = c^m$ con $m \leq n$
- In ciascuna delle casistiche, tuttavia, si ha che $uv^0 wx^0 y = uwy \notin L$, in quanto vengono alterati al massimo due tra i numeri di ripetizioni dei tre tipi di carattere a, b e c.

PROPRIETÀ DEI LINGUAGGI LIBERI DAL CONTESTO

Teorema. *I linguaggi CF sono chiusi rispetto all'unione, alla concatenazione, ed alla chiusura di Kleene.*

Dimostrazione:

- Siano $G_1 = \langle V_1, T_1, P_1, S_1 \rangle$ e $G_2 = \langle V_2, T_2, P_2, S_2 \rangle$ le grammatiche generanti i linguaggi L_1 e L_2.
- Per semplicità si assuma che V_1 e V_2 siano disgiunti (altrimenti si proceda a ridenominare le variabili).
- Sia S una nuova variabile.
- $G_\cup = \langle V_1 \cup V_2 \cup \{S\}, T_1 \cup T_2, P_1 \cup P_2 \cup P, S \rangle$, ove P consta delle produzioni:

  $$S \to S_1 | S_2$$

  è la grammatica che genera $L_1 \cup L_2$.

PROPRIETÀ DEI LINGUAGGI LIBERI DAL CONTESTO (CONT.)

- $G_\circ = \langle V_1 \cup V_2 \cup \{S\}, T_1 \cup T_2, P_1 \cup P_2 \cup P, S\rangle$, ove P consta della produzione:

$$S \to S_1 S_2$$

  è la grammatica che genera $L_1 L_2$.

- $G_* = \langle V_1 \cup \{S\}, T_1, P_1 \cup P, S\rangle$, ove P consta delle produzioni:

$$S \to \varepsilon | S_1 S$$

  è la grammatica che genera L_1^*.

PROPRIETÀ DEI LINGUAGGI LIBERI DAL CONTESTO (CONT.)

Teorema. *I linguaggi CF non sono chiusi rispetto all'intersezione.*

Dimostrazione:

➜ Consideriamo $L_1 = \{ a^i b^i c^j \mid i \geq 1, j \geq 1 \}$ generato da:

$$S \to RC, \quad R \to ab \,|\, aRb, \quad C \to c \,|\, cC$$

e $L_2 = \{ a^i b^j c^j \mid i \geq 1, j \geq 1 \}$ generato da:

$$S \to AR, \quad R \to bc \,|\, bRc, \quad A \to a \,|\, aA$$

➜ La loro intersezione è $\{ a^i b^i c^i \mid i \geq 1 \}$, che sappiamo non essere CF.

PROPRIETÀ DEI LINGUAGGI LIBERI DAL CONTESTO (CONT.)

Corollario 5. *I linguaggi CF non sono chiusi rispetto alla complementazione.*

Dimostrazione: Se lo fossero, allora lo sarebbero anche rispetto all'intersezione, in quanto $A \cap B = \overline{\overline{A} \cup \overline{B}}$, ma sappiamo che non lo sono.

ALGORITMI DI DECISIONE PER LINGUAGGI CF

Teorema (Vuoto, Finito, Infinito)**.** *Data una grammatica CF* $G = \langle V, T, P, S \rangle$*, i problemi*

1. $L(G) = \varnothing$*,*
2. $L(G)$ *è finito, e*
3. $L(G)$ *è infinito*

sono decidibili.

Dimostrazione di 1: $L(G) = \varnothing$ se e solo se l'algoritmo del lemma che mostra come eliminare i simboli improduttivi termina con V' tale che $S \notin V'$.

ALGORITMI DI DECISIONE PER LINGUAGGI CF (CONT.)

Dimostrazione di 2 e 3:

- ➜ Se $L(G) = \varnothing$ oppure $L(G) = \{\varepsilon\}$ il linguaggio è finito.
- ➜ Altrimenti, sia $G' = \langle V', T, P', S\rangle$ una grammatica in forma normale di Chomsky, priva di simboli inutili e tale che $L(G') = L(G) \setminus \{\varepsilon\}$.
- ➜ Creiamo un grafo $\mathcal{G}$ avente:
  - un nodo per ogni variabile $A \in V'$;
  - gli archi $\langle A, B\rangle$ e $\langle A, C\rangle$ per ogni produzione $A \to BC \in P'$.

  Allora $L(G)$ (e anche $L(G')$) è finito se e solo se $\mathcal{G}$ non ha cicli.
- ➜ Se $\mathcal{G}$ non ha cicli, la finitezza è immediata: solo un numero finito di alberi di derivazione con radice etichettata da S possono essere costruiti.
- ➜ Viceversa, supponiamo $\mathcal{G}$ abbia (almeno) un ciclo passante per un nodo associato ad una variabile A. In tal caso, ripetendo la costruzione vista nella dimostrazione del pumping lemma, si riescono a generare infiniti alberi di derivazione diversi.

ALGORITMI DI DECISIONE PER LINGUAGGI CF (CONT.)

Teorema (Appartenenza)**.** *Data una grammatica CF* $G = \langle V, T, P, S \rangle$*, e una stringa* z*, il problema* $z \in L(G)$ *è decidibile.*

Dimostrazione:

- Per il caso $z = \varepsilon$, dal Teorema di eliminabilità delle ε-produzioni si ha che $\varepsilon \in L$ se e solo se esiste $S \to A_1 \cdots A_n$ in P (con $n \geq 0$) tale che $A_i \in N$ per $i = 1, \ldots, n$.
- Sia $z \neq \varepsilon$ e $G' = \langle V', T, P', S \rangle$ la grammatica equivalente a G in forma normale di Greibach.
- Allora, se z ha una derivazione, ne ha una di esattamente $|z|$ passi. Generiamo esaustivamente tutte le derivazioni di $|z|$ passi e verifichiamo se esiste una di queste che deriva z.
- Nota bene: esistono algoritmi più efficienti.

ALGORITMI DI DECISIONE PER LINGUAGGI CF (CONT.)

Il seguenti problemi sui linguaggi acontestuali sono indecidibili:

- date G_1 e G_2, dire se $L(G_1) = L(G_2)$ (vedi Teorema 8.12 in *Hopcroft & Ullmann*, 1979);
- date G_1 e G_2, dire se $L(G_1) \cap L(G_2) = \varnothing$;
- date G_1 e G_2, dire se $L(G_1) \subseteq L(G_2)$;
- data G, dire se G è ambigua;
- data $G = \langle V, T, P, S \rangle$, dire se $L(G) = T^*$.

GRAMMATICHE REGOLARI

Una grammatica CF si dice lineare destra se ogni produzione è della forma:

$$A \to wB, \quad \text{con } w \in T^+, \text{ oppure}$$
$$A \to w, \quad \text{con } w \in T^+$$

più eventualmente $S \to \varepsilon$. Se invece tutte le produzioni sono della forma:

$$A \to Bw, \quad \text{con } w \in T^+, \text{ oppure}$$
$$A \to w, \quad \text{con } w \in T^+$$

più eventualmente $S \to \varepsilon$, si dice lineare sinistra.

GRAMMATICHE REGOLARI (CONT.)

Lemma 8. *Data una grammatica lineare destra G esiste una grammatica G' equivalente (in forma normale di Greibach) tale che tutte le produzioni sono della forma:*

$$\begin{aligned} A &\to aB, \quad \textit{con } a \in T, \textit{ oppure} \\ A &\to a, \quad \textit{con } a \in T \end{aligned}$$

più eventualmente $S \to \varepsilon$.

Dimostrazione:

- ➜ Ogni produzione della forma $A \to a_1 a_2 \cdots a_n B$ viene sostituita dalle produzioni: $A \to a_1 C_1$, $C_1 \to a_2 C_2$, ..., $C_{n-1} \to a_n B$, dove $C_1, \ldots, C_{n-1}$ sono nuovi simboli non terminali.
- ➜ Similmente, $A \to a_1 a_2 \cdots a_n$ viene sostituita dalle produzioni: $A \to a_1 C_1$, $C_1 \to a_2 C_2$, ..., $C_{n-1} \to a_n$.

GRAMMATICHE REGOLARI (CONT.)

Teorema. *Se L è generato da una grammatica lineare destra, allora L è un linguaggio regolare.*

Dimostrazione:

- ➜ Sia $G = \langle V, T, P, S\rangle$ tale che $L(G) = L$ e nella forma semplificata descritta nel precedente lemma.
- ➜ Costruiamo un NFA $M = \langle Q, \Sigma, \delta, q_0, F\rangle$ che riconosce L:
  - $Q = V \cup \{\bot\}$;
  - $\Sigma = T$;
  - $B \in \delta(A, a)$ sse $A \to aB \in P$, $\bot \in \delta(A, a)$ sse $A \to a \in P$; $\delta(\bot, a)$ è indefinito per ogni simbolo $a \in \Sigma$.
  - $q_0 = S$;
  - $F = \begin{cases} \{\bot, S\}, & \text{se } S \to \varepsilon \in P; \\ \{\bot\}, & \text{altrimenti.} \end{cases}$

GRAMMATICHE REGOLARI (CONT.)

- Si deve mostrare che $\hat{\delta}(S, w) \cap F \neq \varnothing$ se e solo se $S \overset{G}{\Rightarrow}_* w$, per ogni stringa w.
- Mostriamo, per induzione su $|w| \geq 0$, che per ogni $A \in V$

$$A \in \hat{\delta}(S, w) \qquad \Longleftrightarrow \qquad S \overset{G}{\Rightarrow}_* wA$$

Base: Sia $w = \varepsilon$. Per definizione, $\hat{\delta}(S, \varepsilon) = \{S\}$. Per la forma della grammatica $S \overset{G}{\Rightarrow}_* \varepsilon A$ se e solo se $S = A$.

Passo: Sia $w = va$.

$$\begin{aligned}
A \in \hat{\delta}(S, va) &\iff A \in \textstyle\bigcup_{B \in \hat{\delta}(S,v)} \delta(B, a) && \text{[def. di } \hat{\delta} \text{ nei NFA]} \\
&\iff A \in \textstyle\bigcup_{B \,:\, S \overset{G}{\Rightarrow}_* vB} \delta(B, a) && \text{[ip. induttiva]} \\
&\iff S \overset{G}{\Rightarrow}_* vaA && \text{[def. di } \delta \ (B \to aA \in P)\text{]}
\end{aligned}$$

GRAMMATICHE REGOLARI (CONT.)

➜ Abbiamo mostrato che, per ogni $w \in T^*$ e per ogni $A \in V$

$$A \in \hat{\delta}(S, w) \qquad \Longleftrightarrow \qquad S \overset{G}{\Rightarrow}_* wA$$

➜ Da questo risultato, per definizione di F,

$$F = \begin{cases} \{\bot, S\}, & \text{se } S \to \varepsilon \in P, \\ \{\bot\}, & \text{altrimenti,} \end{cases}$$

si ha che:

1. $S \overset{G}{\Rightarrow}_* \varepsilon$ se e solo se $\hat{\delta}(S, \varepsilon) \cap F \neq \varnothing$;
2. $S \overset{G}{\Rightarrow}_* va$ se e solo se esiste A tale che $S \overset{G}{\Rightarrow}_* vA$ e $A \to a \in P$. Ciò accade se e solo se $A \in \hat{\delta}(S, v)$ e $\bot \in \delta(A, a)$, ovvero $\bot \in \hat{\delta}(S, va)$.

GRAMMATICHE REGOLARI (CONT.)

Teorema. *Se L è un linguaggio regolare, allora L è generato da una grammatica lineare destra.*

Dimostrazione:

- ➜ Sia $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ il DFA che riconosce L.
- ➜ Costruiamo una grammatica lineare destra $G = \langle V, T, P, S \rangle$ tale che $L(G) = L$ nel modo seguente:
  - $V = Q$;
  - $T = \Sigma$;
  - $$\begin{cases} P \ni A \to aB & \iff \delta(A, a) = B \\ P \ni A \to \varepsilon & \iff A \in F \end{cases}$$
  - $S = q_0$.

GRAMMATICHE REGOLARI (CONT.)

- Si mostra per induzione su $|w|$ che $\hat{\delta}(q_0, w) = p$ se e solo se $q_0 \overset{G}{\Rightarrow}_{|w|} wp$ (farlo per esercizio).
- A questo punto si ha immediatamente che $\hat{\delta}(q_0, w) \in F$ se e solo se $q_0 \overset{G}{\Rightarrow}_{|w|+1} w$.
- Per costruzione, la grammatica ottenuta può avere ε-produzioni e dunque non è della forma desiderata. Si applichi dunque l'eliminazione di tali ε-produzioni per ottenere la grammatica nella forma voluta.

GRAMMATICHE REGOLARI (CONT.)

- Il teorema appena visto implica che ogni linguaggio regolare è anche CF.
- Si possono mostrare risultati analoghi a quelli visti per le grammatiche lineari destre anche per le grammatiche lineari sinistre (che quindi sono del tutto equivalenti).

GRAMMATICHE DI TIPO 0

Le grammatiche di tipo 0, o a struttura di frase, sono le grammatiche della forma $G = \langle V, T, P, S \rangle$ in cui P contiene produzioni

$$\alpha \to \beta$$

dove $\alpha \in (V \cup T)^+, \beta \in (V \cup T)^*$.

Teorema. *Si ha $L = L(G)$ con G grammatica a struttura di frase se e solo se L è un insieme ricorsivamente enumerabile.*

GRAMMATICHE DI TIPO 1

Le grammatiche di tipo 1, o dipendenti dal contesto o monotone, sono le grammatiche della forma $G = \langle V, T, P, S \rangle$ in cui P contiene produzioni

$$\alpha \to \beta$$

ove $\alpha \in (V \cup T)^+, \beta \in (V \cup T)^+$, e $|\alpha| \leq |\beta|$, con l'eccezione dell'eventuale produzione $S \to \varepsilon$ (in tal caso richiediamo che S non occorra mai in β).

Per queste grammatiche esiste una forma normale per le produzioni, che devono essere del tipo:

$$\alpha_1 A \alpha_2 \to \alpha_1 \beta \alpha_2$$

con $\beta \neq \varepsilon$.

GRAMMATICHE DI TIPO 1 (CONT.)

Teorema. *Ogni linguaggio L generato da una grammatica dipendente dal contesto è ricorsivo.*

Teorema. *Ci sono insiemi ricorsivi che non sono generati da nessuna grammatica dipendente dal contesto.*

GRAMMATICHE DI TIPO 1 (CONT.)

➜ Il linguaggio $\{ a^i b^i c^i \mid i \geq 1 \}$ è generato dalla grammatica di tipo 1:

$$\begin{array}{rcl} S & \to & aSBC \mid aBC \\ CB & \to & BC \\ bB & \to & bb \end{array} \qquad \begin{array}{rcl} bC & \to & bc \\ cC & \to & cc \\ aB & \to & ab \end{array}$$

➜ Tutte le derivazioni iniziano con $S \overset{G}{\Rightarrow}_* aaa \cdots aBCBCBC \cdots BC$. Le B e le C tengono traccia del numero di b e c da inserire.

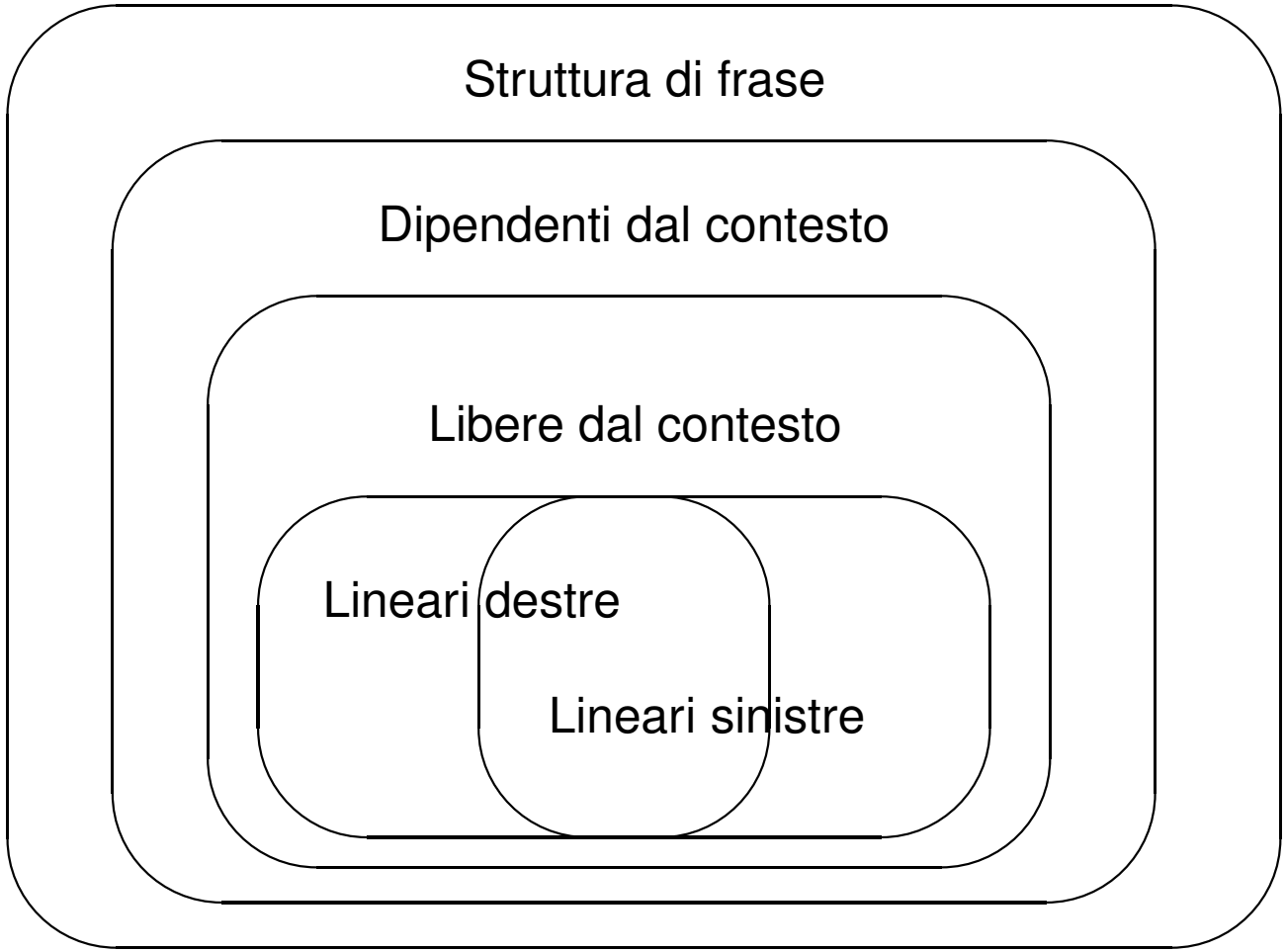
➜ Le tre regole che possono essere usate sono $aB \to ab$ che sistema la b più a sinistra oppure $CB \to BC$, che potremmo definire *regola di scambio*, e la regola $bC \to bc$, che non va applicata prima del tempo ($bCB \overset{G}{\Rightarrow} bcB$).

➜ Invece, con un numero opportuno di applicazioni della regola di scambio, si giunge a a $S \overset{G}{\Rightarrow}_* aaa \cdots abCBC \cdots BC \cdots C$ da cui applicando $bB \to bb$, $bC \to bc$, e $cC \to cc$, si ottiene la stringa attesa.

La Gerarchia di Chomsky

- Sappiamo che il linguaggio $\{ a^i b^i \mid i \geq 0 \}$ è un linguaggio CF ma non regolare.
- Sappiamo inoltre che le grammatiche (CF) lineari destre e sinistre generano esattamente i linguaggi regolari.
- Sappiamo che il linguaggio $\{ a^i b^i c^i \mid i \geq 1 \}$ è un linguaggio dipendente dal contesto ma non CF.
- Dai risultati visti sappiamo anche che la classe di linguaggi generati da grammatiche dipendenti dal contesto è inclusa strettamente nella classe di linguaggi generati da grammatiche a struttura di frase.
- Viene così indotta una gerarchia di grammatiche, nota come gerarchia di Chomsky in cui tutte le inclusioni sono proprie.

LA GERARCHIA DI CHOMSKY (CONT.)



NOZIONE INTUITIVA DI ALGORITMO

a Un algoritmo è di lunghezza finita.

b Esiste un agente di calcolo che porta avanti il calcolo eseguendo le istruzioni dell'algoritmo.

c L'agente di calcolo ha a disposizione una memoria dove vengono immagazzinati i risultati intermedi del calcolo.

d Il calcolo avviene per passi discreti.

e Il calcolo non è probabilistico.

NOZIONE INTUITIVA DI ALGORITMO (CONT.)

f Non deve esserci alcun limite finito alla lunghezza dei dati di ingresso.

g Non deve esserci alcun limite finito alla quantità di memoria disponibile.

h Deve esserci un limite finito alla complessità delle istruzioni eseguibili dal dispositivo.

i Sono ammesse esecuzioni con un numero di passi finito ma illimitato.

NOZIONE INTUITIVA DI ALGORITMO (CONT.)

- ➜ Se ci fermassimo qui, gli algoritmi calcolerebbero solo funzioni totali (definite per ogni input).
- ➜ Ma qualunque formalismo che calcoli solo funzioni totali non le calcola tutte!

NOZIONE INTUITIVA DI ALGORITMO (CONT.)

- Per assurdo, supponiamo che in un certo formalismo si possano calcolare tutte e sole le funzioni totali.
- Sia dunque P_x l'x-esimo programma in questo formalismo e che P_x calcoli la funzione g_x.
- g_x è totale, ma allora anche $h(x) = g_x(x) + 1$ lo è.
- Siccome il nostro formalismo calcola tutte le funzioni totali, avremo $h = g_{x_0}$.
- Ma allora possiamo scrivere

$$g_{x_0}(x_0) = h(x_0) = g_{x_0}(x_0) + 1, \quad \text{assurdo!}$$

NOZIONE INTUITIVA DI ALGORITMO (CONT.)

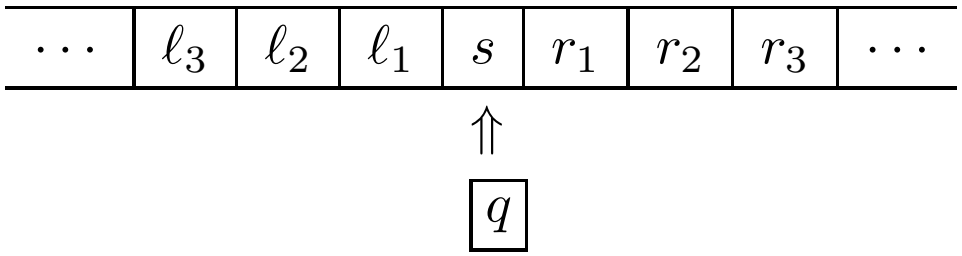
Dobbiamo quindi aggiungere ai punti **a**–**i** già visti il seguente:

l Sono ammesse esecuzioni con un numero di passi infinito (cioè non terminanti).

MACCHINE DI TURING

- Le Macchine di Turing (MdT) sono state ideate da Alan Turing negli anni '30.
- Turing, nella sua analisi del concetto di calcolabilità, introduce per primo la parola computer.
- Questa nozione di computer include tutte le caratteristiche di base dei calcolatori moderni (memoria, input/output, stato) e, come vedremo, modella in modo soddisfacente il concetto di effettiva calcolabilità.

MdT: descrizione modellistica



Il simbolo presente nella casella esaminata dalla testina (s) rappresenta l'input alla macchina. In risposta la macchina può decidere di modificare il simbolo e/o spostare la testina a sinistra o a destra della casella esaminata. Questo permette di avere in input un simbolo diverso per eseguire il passo successivo della computazione (il simbolo ℓ_1 se lo spostamento sarà a sinistra, r_1 se a destra). Inoltre il controllo può cambiare il suo stato (q nel disegno) scegliendolo da un insieme finito di stati possibili.

MdT: matrice funzionale

		s_j	
q_i		$q_r\ s_k\ L$	

Se il controllo è nello stato q_i e la testina legge il simbolo s_j, la macchina:

- si porta in uno stato (interno) q_r;
- scrive il simbolo s_k;
- sposta il nastro a destra, R, o a sinistra, L.

Nel caso $s_j = s_k$, la macchina lascia inalterato il simbolo sul nastro. Se la casella corrispondente alla coppia $\langle s_j, q_i \rangle$ è vuota, la macchina si ferma.

MdT: descrizione formale

Una Macchina di Turing M consiste di:

1. un alfabeto $\Sigma = \{s_0, \ldots, s_n\}$ con almeno due simboli distinti $s_0 = \$$ (blank) e $s_1 = 0$ (tally);
2. un insieme finito di stati $Q = \{q_0, \ldots, q_m\}$ tra i quali vi è lo stato iniziale q_0 (dunque Q è non vuoto);
3. un insieme finito di istruzioni (o quintuple) $P = \{I_1, \ldots, I_p\}$ ognuna delle quali di uno dei seguenti 2 tipi base:
   - $q\ s\ q'\ s'\ R$
   - $q\ s\ q'\ s'\ L$

   tale che non esistono due istruzioni che iniziano con la medesima coppia q, s.

MdT: funzione di transizione

Si può immaginare P come la descrizione "estensionale" (ovvero per casi) di una funzione (in generale) parziale

$$\delta : (Q \times \Sigma) \longrightarrow \big(Q \times \Sigma \times \{R, L\}\big).$$

Tale funzione δ è detta funzione di transizione.

MdT: descrizione istantanea

Una descrizione istantanea (ID) della macchina è una quadrupla

$$\langle q, v, s, w \rangle$$

dove:

- $v, w \in \Sigma^\star$ rappresentano i caratteri significativi presenti sul nastro a sinistra ed a destra della testina;
- s è il simbolo letto dalla testina, essendo la macchina nello stato q.

Ad esempio, se la situazione è

$$\ldots \$ \$ \$ \, s_1 \ldots s_{i-1} \; \underset{s_i}{} \; s_{i+1} \ldots s_j \, \$ \$ \$ \ldots$$

$$\Uparrow$$

$$\boxed{q}$$

allora $v = s_1 \ldots s_{i-1}$, $s = s_i$ e $w = s_{i+1} \ldots s_j$.

MdT: descrizione istantanea

La relazione successore, $\vdash \subseteq ID \times ID$, rappresenta l'esecuzione di un singolo passo di calcolo di una MdT. E' definita da:

- $\langle q, v, r, s\, w\rangle \vdash \langle q', v\, r', s, w\rangle$ se $q\ r\ q'\ r'\ R$ è una istruzione di P;
- $\langle q, v\, s, r, w\rangle \vdash \langle q', v, s, r'\, w\rangle$ se $q\ r\ q'\ r'\ L$ è una istruzione di P.

Una computazione è una sequenza finita di ID $\alpha_0, \ldots, \alpha_n$ tale che

- $\alpha_0 = \langle q_0, v, s, w\rangle$,
- $\alpha_i \vdash \alpha_{i+1}$ per ogni $i \in \{0, \ldots, n-1\}$.

Una computazione è detta terminante se esiste un $n \geq 0$ per cui $\alpha_n = \langle q, v, s, w\rangle$ e non vi è nessuna istruzione di P iniziante con q ed s (cioè $\delta(q, s)$ non è definita), ovvero $\alpha_n \nvdash$.

MdT e nozione intuitiva di algoritmo

a Le istruzioni della MdT sono finite.

b La MdT è l'agente di calcolo che esegue l'algoritmo.

c Il nastro rappresenta la memoria della macchina.

d La MdT opera in modo discreto.

e Ad ogni ID corrisponde una sola azione (determinismo della MdT).

f Non esiste alcun limite all'input, essendo il nastro illimitato.

g La capacità della memoria (nastro) è illimitata.

h Le operazioni eseguibili sono semplici e quindi di complessità limitata.

i Non esiste alcun limite al numero di istruzioni eseguite in quanto la medesima quintupla può essere usata più volte.

l Possono esistere MdT che non calcolano nulla generando una sequenza infinita di ID.

DETERMINISMO DELLE MDT

Se $\alpha \vdash \beta$ e $\alpha \vdash \gamma$ con $\alpha, \beta, \gamma \in ID$, allora $\beta \equiv \gamma$.

Dimostrazione: Ovvio, poiché per definizione non esistono due quintuple $q\ s\ q'\ s'\ X$ aventi la medesima coppia q, s.

ESEMPIO DI MDT: INCREMENTO DI UNO IN BINARIO

Sia $\Sigma = \{\$, 0, 1\}$. Si vuole passare da una configurazione iniziale

$$\ldots \$ \; \$ \; \$ \; s_n \ldots s_0 \; \underset{\Uparrow}{\$} \; \$ \; \$ \; \$ \ldots$$

$$\boxed{q_0}$$

ad una finale

$$\ldots \$ \; \$ \; \$ \; \underset{\Uparrow}{\$} \; s'_m \ldots s'_0 \; \$ \; \$ \; \$ \ldots$$

$$\boxed{q_2}$$

dove

$$\sum_{i=0}^{m} 2^i \cdot s'_i = 1 + \sum_{i=0}^{n} 2^i \cdot s_i.$$

ESEMPIO DI MDT: INCREMENTO DI UNO IN BINARIO (CONT.)

Una possibile definizione della funzione δ sarà la seguente:

δ	\$	0	1
q_0	$q_1\ \$\ L$		
q_1	$q_2\ 1\ L$	$q_2\ 1\ L$	$q_1\ 0\ L$
q_2		$q_2\ 0\ L$	$q_2\ 1\ L$

ESEMPIO DI MDT: COPIA DI UNA STRINGA UNARIA

Sia $\Sigma = \{\$, 0\}$. Si vuole passare da una configurazione iniziale

$$\ldots \$ \$ \$ \underbrace{0 \ldots 0}_{n} \$ \quad \$ \$ \$ \ldots$$

$$\Uparrow$$

$$\boxed{q_0}$$

ad una finale

$$\ldots \$ \$ \$ \underbrace{0 \ldots 0}_{n} \$ \underbrace{0 \ldots 0}_{n} \$ \$ \$ \ldots$$

$$\Uparrow$$

$$\boxed{q_f}$$

ESEMPIO DI MDT: COPIA DI UNA STRINGA UNARIA (CONT.)

Una possibile definizione della funzione δ sarà la seguente:

δ	\$	0
q_0	q_1 \$ L	
q_1		q_2 \$ R
q_2	q_3 \$ R	q_2 0 R
q_3	q_4 0 L	q_3 0 R
q_4	q_5 \$ L	q_4 0 L
q_5	q_6 0 L	q_5 0 L
q_6	q_7 \$ R	q_2 \$ R
q_7		q_7 0 R

Si provi, per esercizio, a definire δ supponendo di disporre di $\Sigma = \{\$, 0, 1\}$.

ESEMPIO DI MDT: INCREMENTO DI UNO IN DECIMALE

Definiamo una MdT che calcola il successore in base 10 di un numero. $S = \{0, \ldots, 9\}$, $Q = \{q_0, q_1\}$ essendo q_1 lo stato di terminazione. La seguente matrice funzionale definisce la MdT desiderata che calcola il successore di un numero scritto sul nastro supponendo che la MdT inizi il calcolo essendo la testina posizionata sulla cifra meno significativa del numero da incrementare.

δ	0	1	2	$\cdots$	7	8	9	$\$$
q_0	$q_1\ 1\ R$	$q_1\ 2\ R$	$q_1\ 3\ R$	$\cdots$	$q_1\ 8\ R$	$q_1\ 9\ R$	$q_0\ 0\ L$	$q_1\ 1\ R$
q_1				$\cdots$				

FUNZIONI CALCOLABILI DA MDT

Una funzione $f : \mathbb{N}^n \longrightarrow \mathbb{N}$ è Turing-calcolabile se esiste una MdT tale che, partendo dalla configurazione iniziale

$$\ldots \$ \$ \$ \quad \underset{\Uparrow}{\$} \quad \underline{x_1} \, \$ \ldots \$ \, \underline{x_n} \, \$ \$ \$ \ldots$$

$$\boxed{q_0}$$

termina nella configurazione

$$\ldots \$ \$ \$ \ldots \$ \, \underline{f(x_1, \ldots, x_n)} \quad \underset{\Uparrow}{\$} \quad \$ \$ \$ \ldots$$

$$\boxed{q_f}$$

se $f(x_1, \ldots, x_n)$ è definita, non termina altrimenti; dove per ogni $(x_1, \ldots, x_n) \in \mathbb{N}^n$, $\underline{x_1}, \ldots, \underline{x_n}, \underline{f(x_1, \ldots, x_n)}$ sono le rappresentazioni in unario rispettivamente di $x_1, \ldots, x_n, f(x_1, \ldots, x_n)$ (mentre $q_f \in Q$ è tale per cui $\delta(q_f, \$)$ è indefinito).

ALCUNE FUNZIONI CALCOLABILI DA MDT

Le seguenti funzioni sono Turing-calcolabili:

- $\lambda\, x.x$: $Q = \{q_0, q_1\}$. δ descritto da:

δ	\$	0
q_0	q_1 \$ R	
q_1		q_1 0 R

- $\lambda\, x.0$: . $Q = \{q_0\}$, δ sempre indefinito.
- $\lambda\, x.x + 1 : Q = \{q_0, q_1\}$. δ definito come segue:

δ	\$	0
q_0	q_1 0 R	
q_1		q_1 0 R

ALCUNE FUNZIONI CALCOLABILI DA MDT (CONT.)

La seguente funzione è Turing-calcolabile:

- $\lambda\, x_1 \ldots x_n.x_i$:

  $Q = \{q_0, \ldots, q_{i+1}\}$. δ definito come segue:

  per $j = 0, \ldots, i$

  $\delta(q_j, 0) = (q_j, 0, R)$;

  $\delta(q_j, \$) = (q_{j+1}, \$, R)$.

MdT generalizzate

Quella che abbiamo visto è una tra le molte possibili variazioni sul tema. Le variazioni che continuano a soddisfare tutti i punti della definizione intuitiva di algoritmo mantengono, però, il medesimo potere espressivo. Valgono, tra gli altri, i seguenti risultati:

- Una MdT con n nastri ed m testine ($m \geq n$) può essere simulata da una MdT con 1 nastro ed una testina;
- Una MdT con n simboli ed m stati può essere simulata da una MdT con 2 stati, aumentando opportunamente il numero dei suoi simboli;
- Una MdT con n simboli ed m stati può essere simulata da una MdT con 2 simboli, aumentando opportunamente il numero dei suoi stati;
- Ogni MdT può essere simulata da una MdT che può solo scrivere e non rimpiazzare simboli sul nastro.

FUNZIONI RICORSIVE DI BASE

- La funzione costante 0: $\lambda x.\ 0$;
- La funzione successore S: $\lambda x.\ x + 1$;
- La funzione identità, o i-esima proiezione di n argomenti, π_i^n: $\lambda x_1, \ldots, x_n.\ x_i$ con $1 \leq i \leq n$.

COMBINATORI

Una funzione $f : \mathbb{N}^n \longrightarrow \mathbb{N}$ si dice:

- definita per composizione da $g_1, \ldots, g_k : \mathbb{N}^n \longrightarrow \mathbb{N}$ e $h : \mathbb{N}^k \longrightarrow \mathbb{N}$ se

$$f(x_1, \ldots, x_n) = h\big(g_1(x_1, \ldots, x_n), \ldots, g_k(x_1, \ldots, x_n)\big);$$

- definita per ricorsione primitiva da $g : \mathbb{N}^{n-1} \longrightarrow \mathbb{N}$ e $h : \mathbb{N}^{n+1} \longrightarrow \mathbb{N}$ se

$$\begin{cases} f(x_1, \ldots, x_{n-1}, 0) = g(x_1, \ldots, x_{n-1}), \\ f(x_1, \ldots, x_{n-1}, y+1) = h\big(x_1, \ldots, x_{n-1}, y, f(x_1, \ldots, x_{n-1}, y)\big). \end{cases}$$

FUNZIONI PRIMITIVE RICORSIVE

La classe $\mathcal{P}$ delle funzioni primitive ricorsive è la più piccola classe (ovvero l'intersezione di tutte le classi) di funzioni

- ➜ contenenti le funzioni ricorsive di base
- ➜ e chiuse per composizione e ricorsione primitiva.

FUNZIONI PRIMITIVE RICORSIVE: ESEMPIO

$$f_1 = \pi_1^1 = \lambda x.\ x$$

$$f_2 = S = \lambda x.\ x + 1$$

$$f_3 = \pi_3^3 = \lambda x_1, x_2, x_3.\ x_3$$

$$f_4 = \lambda x_1, x_2, x_3.\ f_2\big(f_3(x_1, x_2, x_3)\big) = \lambda x_1, x_2, x_3.\ x_3 + 1$$

$$f_5 = \begin{cases} f_5(x_2, 0) & = f_1(x_2) \\ f_5(x_2, y+1) & = f_4\big(x_2, y, f_5(x_2, y)\big) \end{cases}$$

$$f_6 = \lambda x.\ f_5\big(f_1(x), f_1(x)\big)$$

Nota: $f_6 = \lambda x.\ 2x$ e $f_5 = \lambda x, y.\ x + y$.

FUNZIONI PRIMITIVE RICORSIVE: ESEMPI

Zero $0^n = \lambda x_1 \cdots x_n.\ 0$: definibile come $0(\pi_1^n(x_1, \ldots, x_n))$;

Costante $c^n = \lambda x_1 \cdots x_n.c$: definibile come

$$\underbrace{S(\cdots S}_{c}(0^n(x_1, \ldots, x_n)\underbrace{)\cdots)}_{c};$$

Somma

$$\begin{cases} +(x, 0) = x \\ +(x, y+1) = S(+(x, y)) \end{cases}$$

Definita per ricorsione primitiva con:

- f è $+$, di arità 2,
- g è π_1^1,
- $h(x, y, z)$, di arità 3, è la funzione $S\big(\pi_3^3(x, y, z)\big)$.

FUNZIONI PRIMITIVE RICORSIVE: ESEMPI

Moltiplicazione

$$\begin{cases} \cdot(x,0) = 0 \\ \cdot(x,y+1) = +(\cdot(x,y),x)) \end{cases}$$

Potenza

$$\begin{cases} x^0 = S(0) \\ x^{y+1} = \cdot(x^y,x)) \end{cases}$$

Iper-potenza

$$\begin{cases} \mathrm{iper}(x,0) = \pi_1^1(x) \\ \mathrm{iper}(x,y+1) = x^{\mathrm{iper}(x,y)} \end{cases}$$

Si osservi che $\mathrm{iper}(x,y) = x\underbrace{^{x\,\cdots\,x}}_{y}$.

FUNZIONI PRIMITIVE RICORSIVE: ESEMPI

Predecessore

$$\begin{cases} \mathrm{pred}(0) = 0 \\ \mathrm{pred}(y+1) = \pi_1^1(y) \end{cases}$$

Differenza

$$\begin{cases} -(x,0) = \pi_1^1(x) \\ -(x,y+1) = \mathrm{pred}(-(x,y)) \end{cases}$$

Fattoriale

$$\begin{cases} 0! = S(0) \\ (y+1)! = \cdot(S(y), y!) \end{cases}$$

FUNZIONI PRIMITIVE RICORSIVE: ESEMPI

Segno

$$\begin{cases} \mathrm{sg}(0) = 0 \\ \mathrm{sg}(y+1) = S(0) \end{cases}$$

Definiamo inoltre $\overline{\mathrm{sg}}(x) = -(1, \mathrm{sg}(x))$.

Valore assoluto della differenza $|-(x,y)| = +(-(x,y), -(y,x))$;

Minimo e massimo

$$\left\{ \begin{array}{lcl} \min(x,y) & = & -(x, -(x,y)) \\ \max(x,y) & = & +(x, -(y,x)) \end{array} \right\}$$

FUNZIONI PRIMITIVE RICORSIVE: ESEMPI

Divisione intera Assumiamo che $\mathrm{div}(x,0) = \mathrm{mod}(x,0) = 0$.

$$\begin{cases} \mathrm{mod}(0,x) = 0 \\ \mathrm{mod}(y+1,x) = S(\mathrm{mod}(y,x)) \cdot \mathrm{sg}(|-(x, S(\mathrm{mod}(y,x)))|) \end{cases}$$

$$\begin{cases} \mathrm{div}(0,x) = 0 \\ \mathrm{div}(y+1,x) = \mathrm{div}(y,x) + \overline{\mathrm{sg}}(|-(x, S(\mathrm{mod}(y,x)))|) \end{cases}$$

Produttoria

Sia f una funzione ricorsiva primitiva binaria, allora definiamo la funzione $\Pi_{z<y} f(x,z) = \Pi_{z=0}^{y-1} f(x,z)$:

$$\begin{cases} \Pi_{z<0} f(x,z) = S(0) \\ \Pi_{z<y+1} f(x,z) = \cdot(f(x,y), \Pi_{z<y} f(x,z)) \end{cases}$$

FUNZIONI PRIMITIVE RICORSIVE: ESEMPI

Sommatoria

Sia f una funzione ricorsiva primitiva binaria, allora definiamo la funzione $\Sigma_{z<y} f(x,z) = \Sigma_{z=0}^{y-1} f(x,z)$:

$$\begin{cases} \Sigma_{z<0} f(x,z) = 0 \\ \Sigma_{z<y+1} f(x,z) = +(f(x,y), \Sigma_{z<y} f(x,z)) \end{cases}$$

FUNZIONI PRIMITIVE RICORSIVE: ESEMPI

μ-operatore limitato di minimizzazione

Sia f una funzione primitiva ricorsiva $n+1$ aria.

$$
\begin{aligned}
g(x_1,\ldots,x_n,y) &= \mu z < y.\,(f(x_1,\ldots,x_n,z)=0) \\
&= \begin{cases} \text{il più piccolo } z \text{ minore di } y & \\ \text{tale che } f(x_1,\ldots,x_n,z)=0 & \text{se tale } z \text{ esiste} \\ y & \text{altrimenti} \end{cases}
\end{aligned}
$$

Tale funzione è primitiva ricorsiva:

$$g(x_1,\ldots,x_n,y) = \Sigma_{v<y}\Big(\Pi_{u\leq v}\mathrm{sg}\big(f(x_1,\ldots,x_n,u)\big)\Big).$$

CI SONO FUNZIONI CALCOLABILI NON PRIMITIVE RICORSIVE

- Le funzioni primitive ricorsive sono totali.
- La funzione di Ackermann è calcolabile, totale, ma non primitiva ricorsiva:

$$\begin{cases} \mathrm{ack}(0,0,y) = y \\ \mathrm{ack}(0,x+1,y) = \mathrm{ack}(0,x,y)+1 \\ \mathrm{ack}(1,0,y) = 0 \\ \mathrm{ack}(z+2,0,y) = 1 \\ \mathrm{ack}(z+1,x+1,y) = \mathrm{ack}\big(z,\mathrm{ack}(z+1,x,y),y\big). \end{cases}$$

FUNZIONI PARZIALI RICORSIVE

- Sia f una funzione totale $n+1$-aria, e sia

$$Z(x_1, \ldots, x_n) = \left\{ z \in \mathbb{N} \;\middle|\; f(x_1, \ldots, x_n, z) = 0 \right\}$$

Definiamo la funzione $\varphi \colon \mathbb{N}^n \to \mathbb{N}$ tale che

$$\varphi(x_1, \ldots, x_n) \uparrow, \qquad \text{se } Z(x_1, \ldots, x_n) = \varnothing;$$
$$\varphi(x_1, \ldots, x_n) = \min Z(x_1, \ldots, x_n), \qquad \text{altrimenti.}$$

- Notazione: la funzione φ si denota con $\mu z \,.\, \big(f(x_1, \ldots, x_n, z) = 0\big)$.
- L'operatore che mappa f in $\varphi = \mu z \,.\, \big(f(x_1, \ldots, x_n, z) = 0\big)$ è detto μ-operatore e φ è detta essere definita per minimizzazione (o μ-ricorsione) da f.

FUNZIONI PARZIALI RICORSIVE (CONT.)

- L'ipotesi che f sia una funzione totale è importante!
- Questa ipotesi può essere rilasciata definendo la minimizzazione diversamente.
- Sia ψ una funzione parziale; φ è definita per μ-ricorsione da ψ se

$$\varphi(x_1, \ldots, x_n) = \mu z \,.\, \big(\forall y \leq z : \psi(x_1, \ldots, x_n, y) \downarrow \wedge \psi(x_1, \ldots, x_n, z) = 0\big).$$

ENTRAMBE LE SPECIFICHE SONO CALCOLABILI

E sono calcolabili nello stesso modo, ad esempio:

```
nat phi(nat x1, ..., nat xn) {
  nat z = 0;
  while (true) {
    if (f(x1, ..., xn, z) == 0)
      return z;
    ++z;
  }
}
```

FUNZIONI PARZIALI RICORSIVE (CONT.)

- La classe delle funzioni parziali ricorsive è la minima classe $\mathcal{PR}$ di funzioni
  - contenente le funzioni ricorsive di base
  - e chiusa per composizione, ricorsione primitiva e μ-ricorsione.
- Esempio: il logaritmo intero si può definire come

$$\lfloor \log_a x \rfloor = \mu y \,.\, \big(\mathrm{leq}(x, a^{y+1}) = 0\big)$$

dove 'leq' è una funzione ricorsiva primitiva tale che $\mathrm{leq}(x, y) = 0$ se e solo se $x > y$; Esempio: la radice n-esima intera si può definire come

-

$$\lfloor \sqrt[n]{x} \rfloor = \mu y \,.\, \Big(\mathrm{leq}\big(x, (y+1)^n\big) = 0\Big).$$

EQUIVALENZA TRA MDT E FUNZIONI PARZIALI RICORSIVE

Teorema. $\varphi\colon \mathbb{N}^n \to \mathbb{N}$ *è parziale ricorsiva se e solo se è Turing-calcolabile.*

Dimostrazione:

$\Longrightarrow$ Si dimostra che se φ è parziale ricorsiva, allora esiste una MdT ("comoda") che la calcola. **(Studiare dettagli sul testo).**

$\Longleftarrow$ Si dimostra che, se $\varphi\colon \mathbb{N} \to \mathbb{N}$ è calcolata da una MdT Z, allora Z è codificabile come funzione parziale ricorsiva. **(Studiare dettagli sul testo).**

Corollario 6 (Forma normale di Kleene)**.** *Per ogni funzione $\varphi \in \mathcal{PR}$ (o equivalentemente Turing calcolabile) esistono $f, g \in \mathcal{P}$ tali che*

$$\varphi = \lambda x \,.\, f\big((\mu t \,.\, g(t,x) = 0), x\big).$$

TESI DI CHURCH-TURING

- Le MdT, le funzioni parziali ricorsive, i sistemi di Post, il λ-calcolo ed altri formalismi sono tutti equivalenti, ...
- ... nel senso che permettono di calcolare esattamente la stessa classe di funzioni.
- Di qui la formulazione nel 1936 da parte di Church e Turing della seguente

  Tesi di Church-Turing: La classe delle funzioni "intuitivamente calcolabili" coincide con la classe delle funzioni Turing calcolabili.

- Si considerano dunque equivalenti i concetti di
  - calcolabilità "effettiva", ovvero associata all'esistenza di una MdT;
  - calcolabilità "ricorsiva", ovvero associata ad una funzione parziale ricorsiva che la calcola;
  - calcolabilità "algoritmica", ovvero associata all'esistenza di un algoritmo che soddisfi i requisiti **a–l**).

ATTENZIONE!

- ➜ Un conto è sapere che esiste un algoritmo che calcola una certa funzione.
- ➜ Altro conto è esibire quel tale algoritmo.

$$f(x) = \begin{cases} 1, & \text{se esattamente } x \text{ '5' consecutivi appaiono nella} \\ & \text{espansione decimale di } \pi; \\ 0, & \text{altrimenti;} \end{cases}$$

$$g(x) = \begin{cases} 1, & \text{se almeno } x \text{ '5' consecutivi appaiono nella} \\ & \text{espansione decimale di } \pi; \\ 0, & \text{altrimenti.} \end{cases}$$

ATTENZIONE! (CONT.)

- La funzione g è chiaramente calcolabile. Infatti, o g è la funzione costante 1, oppure, per un certo $k \in \mathbb{N}$,

$$g(x) = \begin{cases} 1, & \text{se } x \leq k; \\ 0, & \text{se } x > k. \end{cases}$$

- Di f non si sa se sua calcolabile o no (a meno che non siano intervenuti nuovi sviluppi in teoria dei numeri).
- Ma questa variante è certamente calcolabile:

$$f'(x) = \begin{cases} 1, & \text{se esattamente } x \text{ '5' consecutivi appaiono nella} \\ & \text{espansione decimale di } \pi; \\ \uparrow, & \text{altrimenti.} \end{cases}$$

L'algoritmo calcola, una alla volta, le cifre dell'espansione decimale di π e cerca una sequenza di esattamente x '5' consecutivi...

Enumerazione delle MdT

- Sia $\Sigma = \{\$, 0\}$. Quante sono le macchine di Turing con 1 stato?
- Sia $Q = \{q_0\}$. Si tratta riempire la tabella

	$\$$	0
q_0		

  in tutti i modi (significativi) possibili.
- Ogni singola casella può essere
  1. *vuota* (ma per convenzione la rappresentiamo con $\$\ \$\ \$$);
  2. $q_0\ \$\ R$;
  3. $q_0\ \$\ L$;
  4. $q_0\ 0\ R$;
  5. $q_o\ 0\ L$.
- Dunque ci sono esattamente $25 = 5 \times 5$ macchine di Turing distinte ad un solo stato.

ENUMERAZIONE DELLE MDT (CONT.)

- Fissiamo un ordinamento su Σ: poniamo $\$ <_\Sigma 0$.
- Fissiamo un ordinamento su $M = \{L, R\}$: poniamo $L <_M R$.
- Si consideri l'ordinamento lessicografico indotto su $Q \times \Sigma \times M$ e si imponga che $\$\ \$\ \$$ sia minore di qualunque terna $q_0\ s\ m$.
- A questo punto si può definire un ordinamento lessicografico $\lhd$ sulle macchine, considerando le celle della tabella da sinistra a destra (e poi, in seguito, dall'alto in basso):

$$\langle \$\$\$, \$\$\$\rangle \lhd \langle \$\$\$, q_0\$L\rangle \lhd \cdots \lhd \langle \$\$\$, q_0 0R\rangle \lhd \langle q_0\$L, \$\$\$\rangle \lhd \cdots \lhd \langle q_0 0R, q_0 0R\rangle.$$

ENUMERAZIONE DELLE MDT (CONT.)

- Sia $Q = \{q_0, \ldots, q_{n-1}\}$, $\Sigma = \{s_0 = \$, s_1 = 0\}$, $m_0 = L$, $m_1 = R$.
- Si considerino le celle ordinate per righe.
- Si definisca l'ordinamento tra i contenuti di ogni cella nel modo seguente:
  - $\$\,\$\,\$ \prec q_i\ s\ m$ per ogni $q_i \in Q$, $s \in \Sigma$, $m \in \{L, R\}$;
  - $q_{i_1}\ s_{j_1}\ m_{k_1} \prec q_{i_2}\ s_{j_2}\ m_{k_2}$ se $i_1 < i_2$ oppure ($i_1 = i_2$ e $j_1 < j_2$) oppure ($i_1 = i_2$ e $j_1 = j_2$ e $k_1 < k_2$).
- Si ottiene così un ordinamento totale sull'insieme delle macchine di Turing con n stati.
  - Queste sono in tutto

$$(n \cdot 2 \cdot 2 + 1)^{2n} = (4 \cdot n + 1)^{2n}.$$

  - Ci saranno dunque 25 MdT ad uno stato, 9^4 MdT a due stati, 13^6 MdT a tre stati,....

ENUMERAZIONE DELLE MDT (CONT.)

➜ A questo punto si associa un numero naturale (un *indice*) ad ogni macchina di Turing in maniera biunivoca:

$1 \div 25$	MdT ad uno stato
$26 \div 25 + 9^4$	MdT a due stati
$25 + 9^4 + 1 \div 25 + 9^4 + 13^6$	MdT a tre stati
$\ldots$	$\ldots$

all'interno di ogni gruppo, poi ci si basa sulla relazione di ordinamento.

➜ In generale, con un semplice algoritmo, dato $x \in \mathbb{N}$ si può risalire alla MdT x-esima.

ENUMERAZIONE DELLE MDT (ESEMPIO)

Assumendo un ordinamento delle celle per cui la più importante è quella in alto a sinistra, la macchina di Turing n. 4399 ha due stati ($Q = \{q_0, q_1\}$), e la sua funzione di transizione è la seguente:

δ	$\$$	0
q_0	$q_1\ \$\ R$	
q_1		

Si osservi che tale MdT calcola la funzione $\lambda x\ .\ 0$.

ENUMERAZIONE DELLE MDT (CONT.)

- Abbiamo visto una tecnica effettiva di enumerazione delle MdT tra le infinite possibili.
- Un'altra, che si basa sulle proprietà della decomposizione in fattori primi dei numeri naturali, è descritta nel testo.
- In ogni caso, ciò che si fa è associare ad una MdT (e dunque alla funzione parziale che essa calcola) un numero naturale in modo univoco.
- I naturali possono quindi rappresentare sia gli argomenti del calcolo, che le MdT che eseguono il calcolo.
- Dunque le MdT possono operare su MdT.
- Niente di strano o di nuovo: un compilatore è un programma che opera su programmi.

ENUMERAZIONE DELLE MDT (CONT.)

- ➜ Si indica con P_x la MdT di indice x in una data (e fissata) aritmetizzazione;
- ➜ Si indica con φ_x la funzione calcolata dalla macchina P_x.

Teorema. *Ci sono $\aleph_0$ macchine di Turing distinte o funzioni parziali ricorsive, e $\aleph_0$ funzioni ricorsive.*

Dimostrazione: Segue banalmente dal fatto che tutte le funzioni costanti sono ricorsive. Pertanto ci sono almeno $\aleph_0$ funzioni ricorsive.

Teorema. *Ci sono esattamente $\aleph_0$ funzioni calcolabili.*

Dimostrazione: Segue dal teorema precedente per la Tesi di Church-Turing.

ESISTENZA DI FUNZIONI NON CALCOLABILI

Teorema. *Esistono funzioni (totali) $f : \mathbb{N} \longrightarrow \mathbb{N}$ non Turing calcolabili.*

Dimostrazione Sappiamo che, dal Teorema di Cantor, che:

$$\begin{aligned} |\{f : \mathbb{N} \longrightarrow \mathbb{N}\}| &\geq |\{f : \mathbb{N} \longrightarrow \{0,1\}\}| \\ &= |\wp(\mathbb{N})| \\ &> |\mathbb{N}| \\ &\geq |\{\, f : \mathbb{N} \to \mathbb{N} \mid f \text{ è Turing calcolabile}\,\}| \end{aligned}$$

MACCHINA DI TURING UNIVERSALE

- E' possibile costruire una MdT che, data la coppia di naturali (x_1, x_2), simula il calcolo di P_{x_1} sull'input x_2.
- Per far questo:
  - calcola la funzione (tabella) di transizione della macchina P_{x_1};
  - prepara una porzione di nastro che codifica x_2;
  - memorizza lo stato iniziale q_0;
  - si posiziona in corrispondenza della prima cella della codifica di x_2;
  - ...

FUNZIONE PARZIALE UNIVERSALE

Teorema. *Esiste un indice z tale che per ogni x e y,*

$$\varphi_z(x,y) = \begin{cases} \varphi_x(y), & \textit{se } \varphi_x(y) \textit{ è definita;} \\ \uparrow, & \textit{altrimenti.} \end{cases}$$

Dimostrazione:

- Sia P_x l'insieme di istruzioni di una MdT di indice x. Sappiamo che esiste un metodo effettivo per ottenere P_x da x.
- Applichiamo P_x all'input y e, se e quando questa termina, prendiamo il risultato come output di una funzione di 2 argomenti $\psi(x,y)$.
- Dunque:

$$\psi(x,y) = \begin{cases} \varphi_x(y), & \text{se } \varphi_x(y) \text{ converge;} \\ \uparrow, & \text{se } \varphi_x(y) \text{ diverge.} \end{cases}$$

FUNZIONE PARZIALE UNIVERSALE (CONT.)

- Applicando la Tesi di Church-Turing, possiamo concludere che ψ è parziale ricorsiva e pertanto esiste un indice z tale che $\psi = \varphi_z$.
- Chiamiamo ψ funzione parziale universale.

TEOREMA S-M-N

Teorema (s-m-n)**.** *Per ogni coppia di interi* $m, n \geq 1$ *esiste una funzione ricorsiva* s_n^m *di* $m+1$ *variabili tale che per ogni* $x, y_1, \ldots, y_m \in \mathbb{N}$ *abbiamo che:*

$$\lambda z_1 \cdots z_n.\ \varphi_x(y_1, \ldots, y_m, z_1, \ldots, z_n) = \varphi_{s_n^m(x, y_1, \ldots, y_m)}.$$

Dimostrazione: Per esercizio (il caso $m = n = 1$ è sul libro di testo).

➜ Il significato è il seguente: se abbiamo un programma che implementa un certo algoritmo su $m+n$ parametri in input, dati m valori per fissare i primi di questi, esiste sempre un programma "specializzato" che accetta i restanti n parametri in input.

TEOREMA S-M-N IN C

Dati comunque $\texttt{y1}, \ldots, \texttt{ym} \in \mathbb{N}$ e `phi_x` definita come

```
nat phi_x(nat x1, ..., xm, z1, ... zn) {
  /* body */
}
```

si può sempre ottenere, automaticamente, la funzione

```
nat phi_smn(nat z1, ... zn) {
  nat x1 = y1; ... nat xm = ym;
  /* body */
}
```

tale che, per ogni $\texttt{e1}, \ldots, \texttt{en} \in \mathbb{N}$,

$\texttt{phi\_x}(\texttt{y1}, \ldots, \texttt{ym}, \texttt{e1}, \ldots, \texttt{em}) = \texttt{phi\_smn}(\texttt{e1}, \ldots, \texttt{em})$.

PROBLEMI INSOLUBILI

Il problema della terminazione:

> *Esiste una procedura effettiva che, dati x e y, determina se $\varphi_x(y)$ è definita o no?*

No, non esiste.

PROBLEMI INSOLUBILI (CONT.)

Lemma 9. *Non esiste una funzione ricorsiva g tale che per ogni x:*

$$g(x) = \begin{cases} 1, & \textit{se } \varphi_x(x) \downarrow; \\ 0, & \textit{se } \varphi_x(x) \uparrow. \end{cases}$$

Dimostrazione:

- Supponiamo per assurdo che esista una MdT di indice i_0 tale che $g = \varphi_{i_0}$. Allora possiamo definire la funzione:

$$g'(x) = \begin{cases} \uparrow, & \text{se } g(x) = 1 = \varphi_{i_0}(x); \\ 0, & \text{se } g(x) = 0 = \varphi_{i_0}(x). \end{cases}$$

- Anche g' sarebbe quindi calcolabile ed esisterebbe dunque i_1 tale che $\varphi_{i_1} = g'$.

PROBLEMI INSOLUBILI (CONT.)

➜ Ma allora:

$$\begin{array}{lllllll} \varphi_{i_1}(i_1)\downarrow & \Leftrightarrow & g'(i_1)=0 & \Leftrightarrow & g(i_1)=0 & \Leftrightarrow & \varphi_{i_1}(i_1)\uparrow \\ \varphi_{i_1}(i_1)\uparrow & \Leftrightarrow & g'(i_1)\uparrow & \Leftrightarrow & g(i_1)=1 & \Leftrightarrow & \varphi_{i_1}(i_1)\downarrow \end{array}$$

➜ Assurdo. Dunque un tale indice i_1 non esiste, ovvero non esiste l'indice i_0 che calcoli g, ovvero g non è ricorsiva.

PROBLEMI INSOLUBILI (CONT.)

Teorema. *Non esiste una funzione ricorsiva ψ tale che per ogni x e y:*

$$\psi(x,y) = \begin{cases} 1 & \textit{se } \varphi_x(y) \downarrow \\ 0 & \textit{se } \varphi_x(y) \uparrow \end{cases}$$

Dimostrazione:

- Se esistesse allora esisterebbe un indice x_0 tale che $\varphi_{x_0}(x) = \psi(x,x)$.
- Ma $\psi(x,x)$ non è altro che la funzione $g(x)$ dimostrata non esistere nel lemma appena visto.

PROBLEMI INSOLUBILI (CONT.)

Teorema. *Non esiste una funzione ricorsiva f tale che per ogni x:*

$$f(x) = \begin{cases} 1 & \textit{se } \varphi_x \textit{ è totale} \\ 0 & \textit{se } \varphi_x \textit{ non è totale} \end{cases}$$

Dimostrazione:

- ➜ Supponiamo che esista una tale funzione f.
- ➜ Allora definiamo

$$g(0) = \mu y \,.\, (f(y) = 1),$$
$$g(x+1) = \mu y \,.\, (y > g(x) \wedge f(y) = 1).$$

- ➜ Poiché sappiamo che esistono $\aleph_0$ funzioni ricorsive totali, anche g sarà totale. Per la tesi di Church e questa osservazione è quindi ricorsiva.
- ➜ Definiamo una funzione h tale che $h = \lambda x.\, \varphi_{g(x)}(x) + 1$.

PROBLEMI INSOLUBILI (CONT.)

- Per la definizione di f segue che h è totale. Inoltre, per la Tesi di Church-Turing, h è ricorsiva.
- Sia dunque z_0 l'indice per cui $h = \varphi_{z_0}$ e sia y_0 tale che $g(y_0) = z_0$. Questo indice esiste per definizione di g.
- Allora $h(y_0) = \varphi_{g(y_0)}(y_0) + 1$.
- Ma $\varphi_{g(y_0)}(y_0) = h(y_0)$ per definizione di y_0.
- Poiché h è totale, questa è una contraddizione.

INSIEMI RICORSIVI E RICORSIVAMENTE ENUMERABILI

- Consideriamo un insieme $I \subseteq \mathbb{N}$.
- Ci chiediamo se I è costruibile in modo algoritmico, ovvero se è possibile generare gli elementi di I mediante una funzione che sia effettivamente calcolabile.

Definizione 4. *Un insieme $I \subseteq \mathbb{N}$ è detto ricorsivamente enumerabile (r.e.) se esiste una funzione parziale ricorsiva ψ tale che $I = \mathrm{dom}(\psi)$.*

Definiamo:

- $\mathrm{RE} \subseteq \wp(\mathbb{N})$, la classe degli insiemi ricorsivamente enumerabili;
- $\mathrm{dom}(\varphi_x) = \{ y \in \mathbb{N} \mid \varphi_x(y) \downarrow \}$, essendo φ_x l'x-esima MdT nella numerazione fissata;
- $\mathrm{range}(\varphi_x) = \{ y \in \mathbb{N} \mid \exists z \,.\, \varphi_x(z) \downarrow \wedge \varphi_x(z) = y \}$.

INSIEMI RICORSIVI E RICORSIVAMENTE ENUMERABILI (CONT.)

- Sia $A \in \mathrm{RE}$. La funzione parziale:

$$\psi_A(x) = \begin{cases} 1 & \text{se } x \in A \\ \uparrow & \text{se } x \notin A \end{cases}$$

  è detta funzione semicaratteristica dell'insieme A.
- È chiaro che $A = \mathrm{dom}(\psi_A)$.
- Inoltre, essendo per ipotesi A r.e., la verifica della condizione $x \in A$ è semidecidibile, ovvero se y è tale che $A = \mathrm{dom}(\varphi_y)$,

$$x \in A \iff \varphi_y(x) \downarrow .$$

- Pertanto abbiamo dimostrato che un insieme è detto r.e. sse ha una funzione semicaratteristica parziale ricorsiva.

INSIEMI RICORSIVI E RICORSIVAMENTE ENUMERABILI (CONT.)

Definizione 5. *Un insieme A è detto ricorsivo se esiste una funzione ricorsiva f_A tale che:*

$$f_A(x) = \begin{cases} 1, & \text{se } x \in A; \\ 0, & \text{se } x \notin A. \end{cases}$$

Esempio 1. *I seguenti sono insiemi ricorsivi:*

- *L'insieme $\{0, 2, 4, 6, \ldots\}$ dei numeri pari;*
- *$\mathbb{N}$ e $\varnothing$;*
- *Ogni insieme finito (dimostrare per esercizio);*
- *Ogni insieme A tale che $\bar{A}$ è finito (dimostrare per esercizio).*

INSIEMI RICORSIVI E RICORSIVAMENTE ENUMERABILI (CONT.)

- Se A è ricorsivo allora $A \in \mathrm{RE}$.
- Infatti, basta considerare la funzione parziale, chiaramente ricorsiva

$$\varphi(x) = \begin{cases} 1, & \text{se } f_A(x) = 1; \\ \uparrow, & \text{altrimenti.} \end{cases}$$

- Vi sono dunque $\aleph_0$ insiemi r.e. di cui una parte di uguale cardinalità è ricorsiva.

TEOREMA DI POST

Teorema (Teorema di Post)**.** *Un insieme A è ricorsivo se solo se A e $\bar{A}$ sono r.e.*

Dimostrazione: Sia $A \subseteq \mathbb{N}$ e $\bar{A} = \mathbb{N} \setminus A$.

($\Longrightarrow$) Sia A ricorsivo e f_A la sua funzione caratteristica (totale) ricorsiva. Definiamo:

$$\psi(x) = \begin{cases} 1, & \text{se } f_A(x) = 1; \\ \uparrow, & \text{altrimenti.} \end{cases}$$

- ➜ ψ è chiaramente una funzione parziale ricorsiva.
- ➜ A ha quindi una funzione semicaratteristica parziale ricorsiva.
- ➜ Ma anche $\bar{A}$ ce l'ha: $f_{\bar{A}} = 1 - f_A$.
- ➜ Dunque A e $\bar{A}$ sono r.e.

TEOREMA DI POST (CONT.)

($\Longleftarrow$) Supponiamo A e $\bar{A}$ r.e., con funzioni semicaratteristiche ψ_A e $\psi_{\bar{A}}$. Definiamo:

$$f(x) = \begin{cases} 1, & \text{se } \psi_A(x)\downarrow; \\ 0, & \text{se } \psi_{\bar{A}}(x)\downarrow. \end{cases}$$

- Se $x \in \mathbb{N}$, o $x \in A$ o $x \in \bar{A}$.
- Eseguendo a turno un'istruzione della MdT che calcola ψ_A e un'istruzione della MdT che calcola $\psi_{\bar{A}}$, prima o poi una delle due MdT termina.
- Dunque f è ricorsiva.

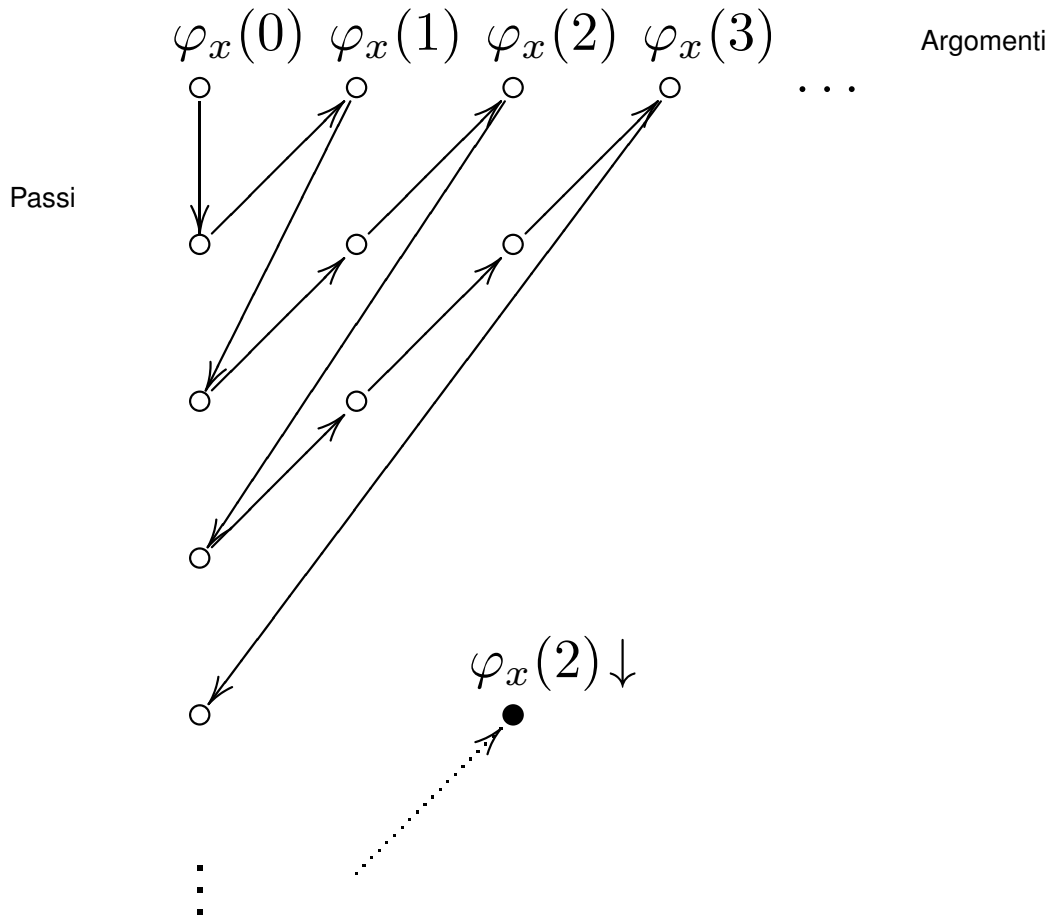
DOVETAILING

Sia φ_x una funzione parziale ricorsiva. Applicare il dovetailing a φ_x significa costruire, in modo induttivo, l'insieme L nel seguente modo:

Passo 0 Si faccia un passo di calcolo per $\varphi_x(0)$, e se termina allora $L := \{\varphi_x(0)\}$;

Passo $n+1$ Si faccia un passo nel calcolo di tutte le funzioni $\varphi_x(0), \ldots, \varphi_x(n+1)$ ove il calcolo non sia già terminato al passo precedente, e per ogni m tale che $\varphi_x(m)$ converge, si pone $L := L \cup \{\varphi_x(m)\}$.

DOVETAILING (CONT.)



L'insieme $L = \mathrm{range}(\varphi_x)$ è stato dunque effettivamente costruito.

TEOREMA DI KLEENE

Teorema (Caratterizzazione degli insiemi r.e.)**.** *Le seguenti affermazioni sono equivalenti:*

1. $A \in \mathrm{RE}$*;*
2. *esiste* $\varphi \in \mathcal{PR}$ *tale che* $A = \mathrm{range}(\varphi)$*;*
3. $A = \varnothing$ *oppure esiste una funzione* $f \in R$ *tale che* $A = \mathrm{range}(f)$*.*

Dimostrazione: Fissiamo un'enumerazione delle MdT (o delle funzioni parziali ricorsive) $\{\varphi_x\}_{x \in \mathbb{N}}$.

TEOREMA DI KLEENE ($1 \Longrightarrow 2$)

- Sia $A \in \mathrm{RE}$.
- Per definizione, esisterà φ_x parziale ricorsiva tale che $A = \mathrm{dom}(\varphi_x)$.
- Definiamo:

$$\delta(y) = \begin{cases} y, & \text{se } \varphi_x(y) \downarrow; \\ \uparrow, & \text{altrimenti.} \end{cases}$$

- δ è parziale ricorsiva e $\mathrm{range}(\delta) = \mathrm{dom}(\varphi_x) = A$.

TEOREMA DI KLEENE ($2 \Longrightarrow 3$)

- Sia $A = \mathrm{range}(\varphi_x)$ e $A \neq \varnothing$.
- Applichiamo il dovetailing a φ_x.
- Poiché $A = \mathrm{range}(\varphi_x) \neq \varnothing$, esiste $n_0 \in \mathbb{N}$ tale che all'n_0-esimo stadio del dovetail si osserva una terminazione.

TEOREMA DI KLEENE ($2 \Longrightarrow 3$, CONT.)

➜ Definiamo la funzione f induttivamente sullo stadio n-esimo del dovetail, in modo tale che f abbia valori tutti e soli nella lista L generata dal dovetail di φ_x.

- Se φ_x converge in $\{m_0, \ldots, m_{j_0}\}$ con $0 \leq j_0 \leq n_0$ allora per ogni $i \in [0, j_0]$ poniamo $f(i) = \varphi_x(m_i)$;
- Supponiamo che per definire $f(j_p)$ si sia usato lo stadio n_p-esimo del dovetail. Possiamo avere i seguenti casi:
  - Se nell'$n_p + 1$-esimo passo del dovetail non si rivelano nuove terminazioni, allora $f(j_p + 1) = f(j_p)$: poniamo $j_{p+1} = j_p + 1$.
  - Se nell'$n_p + 1$-esimo passo del dovetail φ_x converge con input $\{p_0, \ldots, p_k\}$ con $0 \leq k \leq n_p + 1$, allora per ogni $i \in [0, k]$: $f(j_p + 1 + i) = \varphi_x(p_i)$: poniamo $j_{p+1} = j_p + 1 + k$.

➜ La procedura che abbiamo descritto è effettiva e definisce una funzione ricorsiva totale f tale che $A = \mathrm{range}(f)$.

TEOREMA DI KLEENE ($3 \Longrightarrow 1$)

- Se $A = \varnothing$ allora $A = \mathrm{dom}(\lambda x\ .\uparrow)$.
- Se $A = \mathrm{range}(f)$ con $f \in \mathcal{R}$, basta porre:

$$\varphi(x) = \mu z\ .\ \big(|f(z) - x| = 0\big)$$

- Infatti $\varphi \in \mathcal{PR}$ e

$$\varphi(x)\downarrow \iff\ \exists z\ .\ f(z) = x,$$

quindi $A = dom(\varphi)$ che dimostra che A è r.e.

Corollario 7. *Esiste una funzione ricorsiva f tale che* $\mathrm{dom}(\varphi_x) = \mathrm{range}(\varphi_{f(x)})$.

Nota bene: non è detto che $\varphi_{f(x)}$ sia totale, pur essendolo f.

CARATTERIZZAZIONE DEGLI INSIEMI RICORSIVI

Teorema. *Le seguenti affermazioni sono equivalenti:*

1. A *è ricorsivo;*
2. $A = \varnothing$ *oppure* $A = \mathrm{range}(f)$ *con* $f \in \mathcal{R}$ *(monotona) non decrescente.*

Dimostrazione:

($\Longrightarrow$). Sia $A \neq \varnothing$ e A ricorsivo. Poiché $A \subseteq \mathbb{N}$, esiste $a = \min(A)$ minimo valore tra quelli contenuti in A. Definiamo:

$$f(0) = a$$

$$f(n+1) = \begin{cases} n+1, & \text{se } n+1 \in A; \\ f(n), & \text{altrimenti.} \end{cases}$$

f è ricorsiva, non decrescente e $A = \mathrm{range}(f)$.

CARATTERIZZAZIONE DEGLI INSIEMI RICORSIVI ($\Longleftarrow$)

- Se A è finito, allora A è banalmente ricorsivo.
- Supponiamo che A sia infinito e $A = \mathrm{range}(f)$ con f ricorsiva non decrescente.
- Sia $x \in \mathbb{N}$ e $z_x = \mu y \,.\, (f(y) > x)$.
- z_x è ben definito e

$$x \in A \iff x \in \{f(0), \ldots, f(z_x)\}$$

- È quindi decidibile l'appartenenza di x ad A, ovvero A è ricorsivo.

INSIEMI RICORSIVI E RICORSIVAMENTE ENUMERABILI (CONT.)

Teorema. *Ogni insieme r.e. infinito ha un sottoinsieme ricorsivo infinito.*

Dimostrazione:

- Sia $A \subseteq \mathbb{N}$ tale che $A = \mathrm{range}(f)$ con f funzione ricorsiva e $|A| = \aleph_0$.
- Definiamo la funzione g tale che:

$$g(0) = f(0)$$
$$g(n+1) = f(\mu y \,.\, f(y) > g(n)).$$

- $g(n+1)$ è dunque il più piccolo elemento di A più grande di $g(n)$.
- g è crescente, quindi $\mathrm{range}(g)$ è ricorsivo per il teorema di cui sopra.
- Inoltre $\mathrm{range}(g) \subseteq A$, poiché per definizione di g abbiamo $\mathrm{range}(g) \subseteq \mathrm{range}(f) = A$.

INSIEMI RICORSIVAMENTE ENUMERABILI NON RICORSIVI

Definizione 6. *Definiamo* $K = \left\{ x \in \mathbb{N} \;\middle|\; \varphi_x(x) \downarrow \right\}$.

Teorema. K *è r.e. ma non ricorsivo.*

Dimostrazione: Definiamo la funzione ψ_K tale che:

$$\psi_K(x) = \begin{cases} 1, & \text{se } \varphi_x(x) \downarrow; \\ \uparrow, & \text{altrimenti.} \end{cases}$$

ψ_K è parziale ricorsiva e semicaratteristica di K, quindi K è r.e.

Supponiamo, per assurdo, K ricorsivo. Allora $\bar{K}$ sarebbe r.e. Sia y_0 tale che $\bar{K} = \mathrm{dom}(\varphi_{y_0})$. Segue che:

$$y_0 \in \mathrm{dom}(\varphi_{y_0}) = \bar{K} \quad \Leftrightarrow \quad y_0 \in K$$

che è chiaramente assurdo.

IL LINGUAGGIO WHILE

- Un linguaggio ad alto livello Turing-equivalente.
- Permette di manipolare semplici strutture dati simili a quelle impiegate in SCHEME e LISP.
- Questa possibilità consente di evitare l'aritmetizzazione: si potranno quindi considerare direttamente i programmi come dati.
- Questo, è il passo fondamentale per definire il concetto di funzione universale.

LE STRUTTURE DATI DI WHILE

- Si tratta di alberi binari.
- Sia A un insieme finito di atomi, o espressioni elementari, e nil l'albero vuoto.
- L'insieme degli alberi $\mathbb{D}_A$ è definito ricorsivamente come il più piccolo insieme tale che:

$$\text{nil} \in \mathbb{D}_A$$

$$A \subseteq \mathbb{D}_A$$

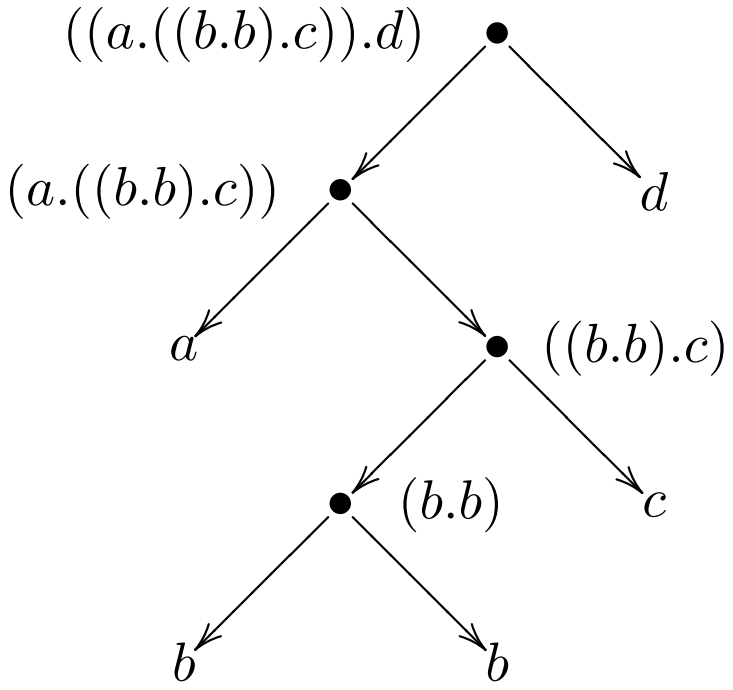
$$\forall d_1, d_2 \in \mathbb{D}_A.\ (d_1.d_2) \in \mathbb{D}_A$$

- In altri termini, se $A = \{a_1, \ldots, a_n\}$, $\mathbb{D}_A$ è il linguaggio generato dalla grammatica acontestuale

$$D_A \longrightarrow \text{nil} \mid a_1 \mid \cdots \mid a_n \mid (D_A.D_A)$$

LE STRUTTURE DATI DI WHILE

Sia $A = \{a, b, c, d\}$. Allora l'albero $((a.((b.b).c)).d)$ si può rappresentare graficamente come



LE STRUTTURE DATI DI WHILE

Teorema. $\mathbb{D}_A$ *è isomorfo a* $\mathbb{N}$.

Dimostrazione: (Traccia) È semplice definire una biiezione tra $\mathbb{N}$ e $\mathbb{D}_A$. Ad esempio si possono prima contare gli alberi di altezza 0, poi quelli di altezza 1 e così via. Ad esempio, se $A = \{a_1, \ldots, a_n\}$:

0	1	2	$\ldots$	n	$n+1$	$n+2$	$\ldots$	n^2+3n+2	n^2+3n+3	$\ldots$
nil	a_1	a_2	$\ldots$	a_n	$(\mathsf{nil}.\mathsf{nil})$	$(\mathsf{nil}.a_1)$	$\ldots$	$(a_n.a_n)$	$(\mathsf{nil}.(\mathsf{nil}.\mathsf{nil}))$	$\ldots$

La sintassi di While

- Sia Var un insieme infinito e numerabile di variabili e sia x una generica variabile.
- Sia d un generico elemento di $\mathbb{D}_A$.
- La sintassi di While è definita dalla seguente grammatica in forma BNF:

$$
\begin{array}{lcl}
Exp & \to & x \mid d \mid \mathsf{cons}(Exp_1, Exp_2) \mid \mathsf{hd}(Exp) \mid \mathsf{tl}(Exp) \mid (Exp_1 = Exp_2) \\
Com & \to & x := Exp \mid Com_1; Com_2 \mid \mathsf{skip} \mid \textbf{while } Exp \textbf{ do } Com \textbf{ endw} \\
Prog & \to & \textbf{read}(Listavar); Com; \textbf{write}(Listavar) \\
Listavar & \to & x \mid x, Listavar
\end{array}
$$

- Si assume inoltre che in $Listavar$ le variabili siano tutte distinte (è un vincolo contestuale!).
- Osservazione: non ci sono dichiarazioni del tipo delle variabili. In While esiste un solo tipo.

LA SINTASSI DI WHILE: ESEMPIO

Il seguente testo è un programma WHILE (darne l'albero di derivazione per esercizio):

read(x);

$y :=$ nil;

while x **do**

$y :=$ cons(hd$(x), y)$;

$x :=$ tl(x)

endw;

write(y)

La semantica di While

- La semantica di un programma While è data in termini di un sistema di transizione.
- Per definirla, dobbiamo prima definire il concetto di stato. Questo rappresenta la memoria della macchina While.
- Ogni cella di memoria corrisponde ad una variabile in Var e
  - contiene un elemento di $\mathbb{D}_A$, oppure
  - è indefinita (in tal caso diremo che contiene il valore speciale $\perp$).
- N.B.: solo un numero finito di celle è definito!
- Dunque l'insieme di tutti gli stati è dato da

$$\mathrm{State} = \mathrm{Var} \to \big(\mathbb{D}_A \cup \{\perp\}\big).$$

- Denoteremo con σ il generico stato.
- $\sigma(x)$ è il valore di x nello stato σ.

La semantica delle espressioni di While

- Ogni espressione rappresenta un albero.
- Al fine di valutare espressioni contenenti variabili è necessario ricorrere alla memoria.
- La semantica delle espressioni è data dunque dalla funzione di interpretazione semantica $\mathcal{E} : (Exp \times State) \to \left(\mathbb{D}_A \cup \{\bot\}\right)$ definita da

$$\mathcal{E}[\![x]\!]\sigma = \sigma(x) \qquad \mathcal{E}[\![d]\!]\sigma = d$$

$$\mathcal{E}[\![\mathsf{cons}(E_1, E_2)]\!]\sigma = (\mathcal{E}[\![E_1]\!]\sigma.\mathcal{E}[\![E_2]\!]\sigma) \qquad \mathcal{E}[\![E_1 = E_2]\!]\sigma = (\mathcal{E}[\![E_1]\!]\sigma = \mathcal{E}[\![E_2]\!]\sigma)$$

$$\mathcal{E}[\![\mathsf{tl}(E)]\!]\sigma = \begin{cases} c & \text{se } \mathcal{E}[\![E]\!]\sigma = (t.c) \\ \mathsf{nil} & \text{altrimenti} \end{cases} \qquad \mathcal{E}[\![\mathsf{hd}(E)]\!]\sigma = \begin{cases} t & \text{se } \mathcal{E}[\![E]\!]\sigma = (t.c) \\ \mathsf{nil} & \text{altrimenti} \end{cases}$$

- Il risultato dell'espressione $E_1 = E_2$ è nil se i due alberi denotati sono diversi, oppure $(\mathsf{nil}.\mathsf{nil})$, se sono uguali.

LA SEMANTICA DEI COMANDI DI WHILE

- Le configurazioni del sistema di transizione sono definite da coppie di $Com \times State$.
- Ogni configurazione $\langle C, \sigma \rangle$ rappresenta il comando C da eseguire e lo stato σ in cui questo viene eseguito.
- E' necessario rappresentare il caso in cui un comando sia stato completamente eseguito. A questo scopo:
  - si rappresenta il comando "vuoto" con ε (il simbolo della stringa vuota);
  - per essere rigorosi si stabilisce che le configurazioni sono, in realtà coppie di $\big(Com \cup \{\varepsilon\}\big) \times \mathrm{State}$;
  - ma, per semplicità notazionale si scrive σ invece di $\langle \varepsilon, \sigma \rangle$.

AGGIORNAMENTO DELLO STATO

➜ Lo stato ottenuto da $\sigma \in \mathrm{State}$ assegnando $d \in \mathbb{D}_A$ ad $x \in \mathrm{Var}$, denotato con $\sigma[d/x]$, è definito come segue, per ogni $x' \in \mathrm{Var}$:

$$\sigma[d/x](x') \stackrel{\mathrm{def}}{=} \begin{cases} d, & \text{se } x' = x; \\ \sigma(x'), & \text{se } x' \neq x. \end{cases}$$

LA SEMANTICA DEI COMANDI DI WHILE

➜ La semantica dei comandi è data dalla relazione $\longrightarrow \subseteq (Com \times State) \times (Com \times State)$ definita da

$$\frac{\mathcal{E}[\![E]\!]\sigma = d}{\langle x := E, \sigma\rangle \longrightarrow \sigma[d/x]} \qquad \frac{}{\langle \mathsf{skip}, \sigma\rangle \longrightarrow \sigma}$$

$$\frac{\langle C_1, \sigma\rangle \longrightarrow \sigma'}{\langle C_1; C_2, \sigma\rangle \longrightarrow \langle C_2, \sigma'\rangle}$$

$$\frac{\mathcal{E}[\![E]\!]\sigma = \mathsf{nil}}{\langle \textbf{while } E \textbf{ do } C \textbf{ endw}, \sigma\rangle \longrightarrow \sigma}$$

$$\frac{\mathcal{E}[\![E]\!]\sigma \neq \mathsf{nil}}{\langle \textbf{while } E \textbf{ do } C \textbf{ endw}, \sigma\rangle \longrightarrow \langle C; \textbf{while } E \textbf{ do } C \textbf{ endw}, \sigma\rangle}$$

LA SEMANTICA DEI PROGRAMMI WHILE

- La semantica di un programma WHILE è definita in termini della chiusura transitiva $\longrightarrow^+$ della relazione di transizione dei comandi.
- Essa definisce una funzione parziale da n-uple di alberi in m-uple di alberi. Questa è codificabile come funzione totale, introducendo (come al solito) il simbolo speciale '$\uparrow$' nel codominio:

$$[\![\cdot]\!]^W : Prog \to (\mathbb{D}_A^n \to \mathbb{D}_A^m \cup \{\uparrow\})$$

- La semantica $[\![P]\!]^W$ di un programma

$$P = \textbf{read}(x_1, \ldots, x_n);\ C;\ \textbf{write}(y_1, \ldots, y_m)$$

è definita, per ogni $d_1, \ldots, d_n \in \mathbb{D}_A$, da

$$[\![P]\!]^W(d_1, \ldots, d_n) = \begin{cases} e_1, \ldots, e_m, & \text{se } \langle C, [d_1/x_1, \ldots, d_n/x_n]\rangle \longrightarrow^* \sigma' \\ & \text{e } \sigma'(y_1) = e_1, \ldots, \sigma'(y_m) = e_m; \\ \uparrow, & \text{altrimenti.} \end{cases}$$

LA SEMANTICA DEI PROGRAMMI WHILE: ESEMPI ED ESERCIZI

Esempio: Il seguente programma calcola l'inversa di una lista rappresentata come un albero sbilanciato a destra.

read(x);

$y := \mathsf{nil}$;

while x **do**

$y := \mathsf{cons}(\mathsf{hd}(x), y)$;

$x := \mathsf{tl}(x)$

endw;

write(y)

LA SEMANTICA DEI PROGRAMMI WHILE: ESERCIZI

Esercizio: Si definisca formalmente il risultato della valutazione di $E_1 = E_2$: $\perp$ se E_1 e/o E_2 valutano a $\perp$; nil se i due alberi denotati sono diversi; $(\mathsf{nil}.\mathsf{nil})$, se sono uguali.

Esercizio: Si tratti a dovere il caso del $\perp$, o, meglio, della propagazione del valore indefinito, nella semantica delle espressioni.

LA SEMANTICA DEI PROGRAMMI WHILE: ESERCIZI

Esercizio: Si fornisca una semantica formale per i comandi:

- if Exp then C_1 else C_2, dal significato intuitivo seguente: si valuti l'espressione Exp; se è diverso da nil si esegua l'istruzione C_1, altrimenti si esegua l'istruzione C_2.
- for $x := Exp$ do C endfor. La semantica intuitiva di questo costrutto è la seguente: si valuta Exp sul valore dello stato iniziale. Essa sarà un albero che necessita di esattamente $n \geq 0$ operazioni di tipo tl per restituire l'espressione nil. L'istruzione C viene quindi ripetuta per n volte; x all'inizio ha il valore iniziale di Exp, poi viene via via ridotto fino a raggiungere nil. x e le variabili che occorrono in Exp non possono essere modificate entro C (questo vincolo è imposto a priori da appositi controlli).

LA SEMANTICA DEI PROGRAMMI WHILE: ESERCIZI

Esercizio: Si mostri, scrivendo frammenti di codice WHILE in cui sono introdotte (se servono) ulteriori variabili, che i due costrutti suddetti possono essere definiti all'interno del linguaggio WHILE. Il che implica che essi non aumentano il potere espressivo di WHILE.

Esercizio: Si mostri come il comando if-then-else dell'esercizio precedente possa essere simulato dal comando for definito nello stesso esercizio. Suggerimento: se α è l'espressione dell'if-then-else, si assegnino le variabili u e v nel seguente modo: $u := (\alpha = \mathsf{nil}); v := (u = \mathsf{nil});$ Si eseguano dunque due cicli for: uno controllato da u che esegue C_2 e uno da v che esegue C_1.

LA SEMANTICA DEI PROGRAMMI WHILE: ESERCIZI

Esercizio: Sia $A = \{a, \ldots, z\}$. Si scriva un Programma WHILE che verifica se un elemento x_1 di A è presente nell'albero x_2 (restituisce $y = (\mathsf{nil}.\mathsf{nil})$ in caso affermativo, $y = \mathsf{nil}$ altrimenti).

ESPRESSIVITÀ DI WHILE

- È possibile rappresentare i numeri naturali in $\mathbb{D}_A$.
- La rappresentazione in $\mathbb{D}_A$ del numero n è denotata $\underline{n}$, ed è definibile come

$$\underline{0} = \mathsf{nil}$$

$$\underline{n+1} = (\mathsf{nil} \,.\, \underline{n})$$

- Osservazione: utilizzando il cons e la sua semantica, si può vedere che anche $\mathsf{cons}(\underbrace{\mathsf{nil}, \mathsf{cons}(\mathsf{nil}, \ldots, \mathsf{cons}(\mathsf{nil}}_{n}, \mathsf{nil}) \cdots)$ rappresenta il numero n.
- Una funzione $f : \mathbb{N}^k \to \mathbb{N}$ è WHILE-calcolabile se esiste un programma WHILE P tale che per ogni $x_1, \ldots, x_k \in \mathbb{N}$:

$$[\![P]\!]^W(\underline{x_1}, \ldots, \underline{x_k}) = \underline{f(x_1, \ldots, x_k)}$$

ESPRESSIVITÀ DI WHILE

➜ Il seguente programma WHILE calcola la funzione $f(x, y) = x + y$:

read(x_1, x_2);
while x_2 **do**
    $x_1 := \mathsf{cons}(\mathsf{nil}, x_1)$;
    $x_2 := \mathsf{tl}(x_2)$
endw;
write(x_1)

Esercizio: Si scrivano i programmi WHILE per il calcolo della differenza tra numeri naturali, del prodotto, delle operazioni `div`, `mod`, e del test $x < y$.

TURING-COMPLETEZZA DI WHILE

Teorema. $f : \mathbb{N}^k \to \mathbb{N}$ *è* WHILE*-calcolabile se e solo se è Turing-calcolabile.*

Dimostrazione: Vedere la traccia sul testo.

GENERALIZZAZIONE AD ALTRI LINGUAGGI DI PROGRAMMAZIONE

- ➜ È possibile generalizzare quanto visto per il linguaggio WHILE ad un arbitrario linguaggio di programmazione.
- ➜ Sia $\mathcal{L}$ un linguaggio di programmazione che definisce funzioni su un insieme di dati $\mathbb{D}$.
- ➜ Supponiamo che esista una codifica univoca dei numeri naturali in $\mathbb{D}$, ovvero per ogni n: $\underline{n} \in \mathbb{D}$ e per ogni $n \neq m$: $\underline{n} \neq \underline{m}$.
- ➜ Sia $[\![\cdot]\!]^{\mathcal{L}}$ la semantica del linguaggio $\mathcal{L}$ definita in modo formale:

$$[\![\cdot]\!]^{\mathcal{L}} : \mathcal{L} \to \big(\mathbb{D} \to (\mathbb{D} \cup \{\uparrow\})\big)$$

- ➜ Una funzione $f : \mathbb{N}^k \to \mathbb{N}$ è $\mathcal{L}$-calcolabile se esiste un programma $P \in \mathcal{L}$ tale che per ogni $x_1, \ldots, x_k \in \mathbb{N}$:

$$[\![P]\!]^{\mathcal{L}}(\underline{x_1}, \ldots, \underline{x_k}) = \underline{f(x_1, \ldots, x_k)}$$

GENERALIZZAZIONE AD ALTRI LINGUAGGI DI PROGRAMMAZIONE

- ➜ Un linguaggio di programmazione $\mathcal{L}$ è detto Turing-completo se l'insieme delle funzioni $\mathcal{L}$-calcolabili coincide con la classe delle funzioni Turing-calcolabili.
- ➜ Il linguaggio WHILE è dunque Turing-completo.
- ➜ Ogni linguaggio di programmazione sufficientemente espressivo per codificare i numeri naturali e comprendente i comandi di base di assegnamento, composizione ed iterazione condizionata indeterminata tipo WHILE è Turing-completo.
- ➜ Infatti, per dimostrare la Turing-completezza di un linguaggio è sufficiente dimostrare che questo è in grado di simulare i costrutti di base di WHILE (o di ogni altro linguaggio Turing-completo).

Esercizio: Una Macchina di Turing si può rappresentare mediante un insieme (lista) di quintuple (lista di 5 elementi). Si scriva un programma While in grado di simulare il comportamento di (ovvero interpretare) una qualsiasi macchina di Turing.

IL LINGUAGGIO FOR

- Abbiamo dimostrato (per esercizio), che il comando **while** è non meno potente dei comandi if-then-else e for.
- Cosa succede se a WHILE togliamo l'iterazione indeterminata data dal **while** ed inseriamo l'if-then-else ed il for?
- Consideriamo il linguaggio FOR la cui sintassi è definita dalla BNF

$$
\begin{array}{lcl}
Exp & \rightarrow & x \mid d \mid \mathsf{cons}(Exp_1, Exp_2) \mid \mathsf{hd}(Exp) \mid \mathsf{tl}(Exp) \mid Exp_1 = Exp_2 \\
Com & \rightarrow & x := Exp \mid Com_1; Com_2 \mid \mathsf{skip} \mid \\
 & & \mathsf{if}\ Exp\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2\ \mathsf{endif} \mid \mathsf{for}\ x := Exp\ \mathsf{do}\ C\ \mathsf{endfor} \\
Prog & \rightarrow & \textbf{read}(x_1, \ldots, x_n); Com; \textbf{write}(y_1, \ldots, y_m)
\end{array}
$$

SEMANTICA DEL LINGUAGGIO FOR

- La semantica di if-then-else e for è quella definita per esercizio.
- Inoltre, come dimostrato per esercizio, il costrutto if-then-else è simulabile dal for e dunque teoricamente superfluo.
- Una funzione $f : \mathbb{N}^k \to \mathbb{N}$ è FOR-calcolabile se esiste un programma FOR P tale che per ogni $x_1, \ldots, x_k \in \mathbb{N}$:

$$[\![P]\!]^F(\underline{x_1}, \ldots, \underline{x_k}) = \underline{f(x_1, \ldots, x_k)}$$

FOR-CALCOLABILITÀ E FUNZIONI PRIMITIVE RICORSIVE

Teorema. $f : \mathbb{N}^m \longrightarrow \mathbb{N}$ *è* FOR*-calcolabile se e solo se* f *è primitiva ricorsiva.*

Dimostrazione: Vedere la traccia sul testo.

Interpreti e Metaprogrammazione

- Per metaprogrammazione si intende la costruzione di programmi che manipolano programmi.
- La MdT universale $\mathbb{U}$ è il prototipo di metaprogramma: essa prende in input l'indice x di una data MdT M ed un dato y, e restituisce in output il risultato ottenibile attivando la macchina M_x sul dato y:

$$\mathbb{U}(x, y) = \begin{cases} \varphi_x(y), & \text{se } \varphi_x(y) \downarrow; \\ \uparrow, & \text{altrimenti.} \end{cases}$$

INTERPRETI E METAPROGRAMMAZIONE

- Traslando questo ragionamento sui linguaggi di programmazione, si ottiene il concetto generale di interprete.
- Siano $\mathcal{L}$ ed $\mathcal{S}$ due linguaggi di programmazione Turing-completi; assumiamo che operino sul medesimo insieme di dati $\mathbb{D}$.
- Un interprete in $\mathcal{L}$ di $\mathcal{S}$-programmi (o semplicemente di $\mathcal{S}$) è un programma $int \in \mathcal{L}$ tale che, per ogni $\mathcal{S}$-programma P, per ogni dato $d \in \mathbb{D}$, e per un'opportuna codifica degli $\mathcal{S}$-programmi in $\mathbb{D}$:
  1. $[\![int]\!]^{\mathcal{L}}(\underline{P}, d) \uparrow$ se e solo se $[\![P]\!]^{\mathcal{S}}(d) \uparrow$;
  2. $[\![int]\!]^{\mathcal{L}}(\underline{P}, d) = [\![P]\!]^{\mathcal{S}}(d)$, se $[\![P]\!]^{\mathcal{S}}(d) \downarrow$.

INTERPRETI E METAPROGRAMMAZIONE

- ➜ L'esistenza di un interprete è assicurata dalla Turing-completezza dei linguaggi in oggetto.
- ➜ Un interprete in $\mathcal{L}$ per $\mathcal{L}$ programmi è detto metainterprete del linguaggio $\mathcal{L}$.

Teorema. *Esiste un interprete in* WHILE *per* WHILE*.*

Dimostrazione: Vedere la traccia sul testo.

METAPROGRAMMAZIONE E INDECIDIBILITÀ

- La possibilità offerta da un linguaggio Turing-completo della metaprogrammazione è alla base della esistenza di problemi algoritmicamente non risolvibili.
- I limiti di ciò che è calcolabile nascono quindi dalle potenzialità del sistema di calcolo stesso.
- Si può dimostrare l'indecidibilità della terminazione utilizzando il linguaggio WHILE.

METAPROGRAMMAZIONE E INDECIDIBILITÀ

➜ Supponiamo esista un programma
$halt \stackrel{\text{def}}{=} \textbf{read}(x_1, x_2); C; \textbf{write}(y) \in$ WHILE, tale che, per ogni $c \in \mathbb{D}$ tale che $c \neq \underline{P}$ per ogni $P \in$ WHILE,

$$[\![halt]\!]^W(c, d) = (\mathsf{nil.nil});$$

inoltre

$$[\![halt]\!]^W(\underline{P}, d) = \begin{cases} (\mathsf{nil.nil}), & \text{se } [\![P]\!]^W(d) \downarrow; \\ \mathsf{nil}, & \text{se } [\![P]\!]^W(d) \uparrow. \end{cases}$$

METAPROGRAMMAZIONE E INDECIDIBILITÀ

➜ Definiamo il programma $R \in$ WHILE nel modo seguente:

read(x);

$x_1 := x; x_2 := x;$

$C;$

while y **do** $y := y$ **endw**;

write(y)

Sia dunque $\underline{R}$ la rappresentazione in $\mathbb{D}$ del programma R.

- Se $[\![R]\!]^W(\underline{R}) \downarrow$, allora l'output di C, che termina sempre per ipotesi, è $y = (\mathsf{nil}.\mathsf{nil})$. Ma allora il programma R entra in loop nel ciclo `while`. Questo implica che $[\![R]\!]^W(\underline{R}) \uparrow$.
- Se $[\![R]\!]^W(\underline{R}) \uparrow$ allora l'output di C è nil. Ma allora il programma R non entra nel ciclo **while** e dunque termina restituendo nil. Dunque $[\![R]\!]^W(\underline{R}) \downarrow$.

METAPROGRAMMAZIONE E INDECIDIBILITÀ

- ➜ Entrambi i casi portano ad un assurdo.
- ➜ Gli interessati possono trovare interessantissimi spunti di riflessione sul rapporto tra indecibilità e capacità introspettive in

  P. Watzlawick, J. H. Beavin, D. D. Jackson,
  Pragmatica della Comunicazione Umana,
  1971, Roma, Astrolabio,
  ISBN 8834001427.