PROLOG E ANALISI SINTATTICA DEI LINGUAGGI

- Quando si vuole definire in modo preciso la sintassi di un linguaggio si ricorre a una *grammatica*

- Una grammatica permette di stabilire se una sequenza di simboli (stringa) dell'alfabeto è una frase del linguaggio

- Convenzione:
  - ε denota la stringa vuota
  - X^+ denota una stringa non vuota costruibile da X
  - $X^* = X^+ \cup \{\varepsilon\}$

PROLOG E ANALISI SINTATTICA DEI LINGUAGGI

- Quando si vuole definire in modo preciso la sintassi di un linguaggio si ricorre a una *grammatica*
- $G=(V_n,V_t,P,S)$
  - dove Vn e' un insieme di *simboli non terminali*
  - Vt e' un insieme di *simboli terminali*
  - P e' un insieme di *produzioni* (o *regole di riscrittura*)

    $\alpha \rightarrow \beta$ scritta anche come $\alpha ::= \beta$

    dove $\alpha \in (V_n \cup V_t)^+$ $\beta \in (V_n \cup V_t)^*$
  - S è lo *scopo* della grammatica (o *simbolo iniziale*)

- Data la produzione $\alpha \rightarrow \beta$ allora per ogni $\mu,\nu \in (V_n \cup V_t)^*$ si dice che $\mu\beta\nu$ è *derivabile* da $\mu\alpha\nu$ in un passo

PROLOG E ANALISI SINTATTICA DEI LINGUAGGI

- Data la produzione $\alpha \rightarrow \beta$ allora per ogni $\mu,\nu \in (V_n \cup V_t)^*$ si dice che $\mu\beta\nu$ è *derivabile* da $\mu\alpha\nu$ in un passo
- La nozione di derivabilità può essere estesa a derivabilità in zero o più passi attraverso la chiusura riflessiva e transitiva ($\rightarrow^*$) della relazione $\rightarrow$.

  $\alpha \rightarrow^* \alpha$

  $\alpha \rightarrow^* \beta$ se $\alpha \rightarrow \gamma$ e $\gamma \rightarrow^* \beta$
- Si definisce linguaggio L(G) individuato dalla grammatica G l'insieme di stringhe formate da simboli terminali derivabili dallo scopo della grammatica

$$L(G)=\{\alpha \mid \alpha \in V_t \text{ e } S \rightarrow^* \alpha\}$$

ESEMPIO

- $G=(V_n,V_t,P,S)$
- $V_n= \{U, V, Z\}$
- $V_t = \{0, 1\}$
- P = { Z ::= U0 | V1
  U ::= Z1 | 1
  V ::= Z0 | 0 }
- P = {01, 0101, 0110, 1010, 10,…. }

Il simbolo | rappresenta la disgiunzione

CLASSIFICAZIONE DELLE GRAMMATICHE

- Tipo 0: *ricorsivamente enumerabile*
  - quando le stringhe che appaiono nella produzioni $\alpha \rightarrow \beta$ non sono soggette ad alcuna limitazione (α e β non vuote)
- Tipo 1: *dipendente dal contesto*
  - quando le produzioni $\alpha A \beta \rightarrow \alpha \pi \beta$ dove A è un non terminale e π è non vuota
- Tipo 2: *libera dal contesto*
  - quando le produzioni sono limitate a $A \rightarrow \pi$ dove A è un non terminale e π è non vuota
- Tipo 3: *regolare (a stati finiti)*
  - quando le produzioni sono limitate a $A \rightarrow a$ o $A \rightarrow aB$ o $A \rightarrow Ba$ dove A e B sono non terminali e a è un terminale

CLASSIFICAZIONE DELLE GRAMMATICHE

- Tipo 3: *regolare (a stati finiti)*
  - quando le produzioni sono limitate a A→ a o A→ aB o A→ Ba dove A e B sono non terminali e a è un terminale

  - Le grammatiche del tipo A→ aB si dicono *lineari a destra*
  - Le grammatiche del tipo A→ Ba si dicono *lineari a sinistra*

$G=(V_n,V_t,P,S)$

V_n= {U, V, Z}

V_t = {0, 1}

P = { Z ::= U0 | V1
U ::= Z1 | 1
V ::= Z0 | 0 }

Grammatica lineare a sinistra

ANALISI SINTATTICA

- Data una grammatica G il problema di verificare se una stringa α appartiene al linguaggio L(G) si dice *analisi sintattica* o *parsing* di α

- Un caso semplice di analisi sintattica è quello dell'analisi dei linguaggi regolari (derivanti da grammatiche di tipo 3). Tale analisi è nota come *analisi lessicale* e può essere risolta da un *automa a stati finiti*

AUTOMI RICONOSCITORI

- Come definire un automa a stati finiti per il riconoscimento di una grammatica regolare sinistra:
  - Esiste uno stato per ogni simbolo non terminale più uno stato iniziale I
  - Lo scopo della grammatica corrisponde allo stato finale
  - Per ogni produzione di tipo A→ Ba c'è un arco dallo stato B allo stato A etichettato con il terminale a
  - Per ogni produzione di tipo A→ a c'è un arco dallo stato iniziale I allo stato A etichettato con il terminale a

AUTOMI RICONOSCITORI

- Come definire un automa a stati finiti per il riconoscimento di una grammatica regolare destra:
  - Esiste uno stato per ogni simbolo non terminale più uno stato finale F
  - Lo scopo della grammatica corrisponde allo stato iniziale
  - Per ogni produzione di tipo A→ aB c'è un arco dallo stato A allo stato B etichettato con il terminale a
  - Per ogni produzione di tipo A→ a c'è un arco dallo stato A allo stato finale F etichettato con il terminale a

ESEMPIO

$G=(V_n,V_t,P,S)$

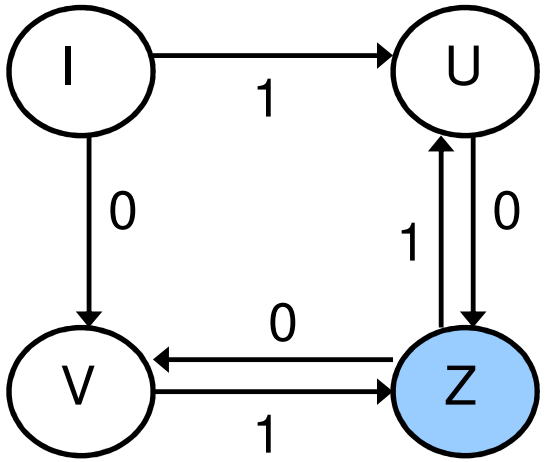
V_n= {U, V, Z}

V_t = {0, 1}

P = { Z ::= U0 | V1

U ::= Z1 | 1

V ::= Z0 | 0 }



Nota:

Z ::= U0 | V1

si scrive come

Z→ U0

Z→ V1

REALIZZAZIONE IN PROLOG

- Per realizzare in Prolog un automa riconoscitore, le transizioni di stato vengono memorizzate da fatti del tipo

  `f(Stato, SimboloIngresso, NuovoStato)`

  e lo stato finale come

  `finale(Stato)`

- Se la stringa da riconoscere è memorizzata in una lista che chiamiamo **`Input`**, la struttura dell'analizzatore è la seguente

  `accept(Input,Stato)`

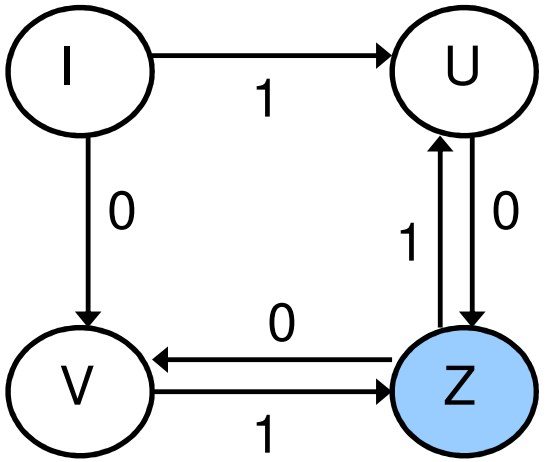
  dove **`Stato`** è lo stato corrente

REALIZZAZIONE IN PROLOG

```
accept([],Stato):- finale(Stato).
accept([Simbolo|Input],Stato):-
   f(Stato, Simbolo, NuovoStato),
   accept(Input, NuovoStato).
```

- Esempio

```
f(u,0,z). f(z,0,v). f(i,0,v).
f(i,1,u). f(v,1,z). f(z,1,u).
finale(z).
```




```
:-accept([0101],i).
yes
```

AUTOMI RICONOSCITORI

- La realizzazione di automi riconoscitori in Prolog per grammatiche regolari è semplice.
- Le grammatiche regolari definiscono linguaggi poco espressivi. Una classe di grammatiche importanti è quella delle grammatiche libere da contesto *context free* perché sono alla base dei linguaggi di programmazione.
- Il problema dell'analisi sintattica per le grammatiche context free è un problema decidibile:
  - si può sempre tradurre una grammatica libera da contesto in un insieme di clausole Prolog e usare la risoluzione per effettuare l'analisi sintattica.

ANALISI SINTATTICA IN PROLOG

- Consideriamo l'analisi sintattica di tipo *top-down*
  - Questo metodo, a partire dallo scopo della grammatica, usa le produzioni come regole di riscrittura e cerca di derivare una stringa uguale alla stringa da riconoscere.

- Vediamo due metodi top down:
  - Metodo dell'automa a pila (stack)
    - simulando il comportamento di un automa a pila, vengono analizzate le produzioni applicabili dal simbolo affiorante sullo stack
  - Metodo ricorsivo discendente
    - simula una derivazione left most attraverso procedure ricorsive, ciascuna della quali corrisponde a una delle produzioni relative a un simbolo non terminale

ANALISI SINTATTICA IN PROLOG

- NOTA: i metodi top-down suddetti non sono applicabili al caso di grammatiche ricorsive a sinistra perché possono entrare in un ciclo infinito.

$$X \rightarrow^* X\alpha \qquad \alpha \in V_n^*$$

  - E' necessaria quindi una trasformazione che elimina la ricorsione a sinistra.

METODO DELL'AUTOMA A PILA

- Utilizziamo due termini **`sn/1`** e **`st/1`** per denotare i simboli non terminali e terminali
- Le regole di produzione sono definite da fatti del tipo **`rule(<non terminale>,<parte destra>).`**

- Se la stringa da riconoscere è memorizzata in una lista che chiamiamo **`Input`**, la struttura dell'analizzatore è la seguente

  `predict(Input,Stack)`

  dove **`Stack`** è lo stack corrente in cui inizialmente viene posto lo scopo della grammatica

METODO DELL'AUTOMA A PILA

```
predict([],[]).
predict(Input, [sn(N)|Stack]):-
    rule(sn(N), ParteDestra),
    append(ParteDestra,Stack,NewStack),
    predict(Input,NewStack).
predict([N|Input], [st(N)|Stack]):-
    predict(Input,Stack).
```

METODO DELL'AUTOMA A PILA

- Esempio:

G = (V_n, V_t, P,S)

V_n = {E, T}

V_n = { +, a }

P = { E::= T + E | T

T::=a}

S = E

```
rule(sn(E),[sn(T),st(+),sn(T)]).
rule(sn(E),[sn(T)]).
rule(sn(T),[st(a)]).


:- predict([a,+,a,+,a],[sn(e)]).
yes
:- predict([a,+,+,a],[sn(e)]).
no
```

METODO RICORSIVO DISCENDENTE

- Le regole di produzione coincidono con clausole Prolog e i simboli terminali e non terminali con predicati.
  - Distinzione tra terminali e non terminali non è più necessaria

- Gli argomenti dei predicati corrispondenti a ogni simbolo x sono: una lista di simboli in ingresso e una lista di simboli in uscita
  - la lista di uscita è una sottolista della lista di ingresso a cui sono stati eliminati
    - gli elementi derivati a partire dal non terminale x, applicando le opportune produzioni
    - x stesso se è un terminale.

METODO RICORSIVO DISCENDENTE

- Esempio:

$G = (V_n, V_t, P, S)$

$V_n = \{E, T\}$

$V_n = \{ +, a \}$

$P = \{ E ::= T + E \mid T$

$T ::= a\}$

$S = E$

```
e(ListIn,ListOut):-
      t(ListIn,ListTemp),
      piu(ListTemp,ListTemp1),
      e(ListTemp1,ListOut).
e(ListIn,ListOut):-
      t(ListIn,ListOut).
t(ListIn,ListOut):-
      a(ListIn,ListOut).
a([a|List],List).
piu([+|List],List).

:- e([a,+,a,+,a],[]).
yes

:- e([a,+,+,a],[]).
no
```

METODO RICORSIVO DISCENDENTE

- Esempio:

$G = (V_n, V_t, P, S)$

$V_n = \{E, T\}$

$V_n = \{ +, a \}$

$P = \{ E ::= T + E \mid T$

$T ::= a\}$

$S = E$

Versione più efficiente

```
e([a,+|ListIn],ListOut):- !,
      e(ListIn,ListOut).
e([a],[]).

:- e([a,+,a,+,a],[]).
yes

:- e([a,+,+,a],[]).
no
```

DEFINITE CLAUSE GRAMMARS (DCG)

- Le *Definite Clause Grammars* sono un'estensione delle grammatiche libere da contesto e rappresentano *descrizioni eseguibili* di una grammatica
- La semantica delle DCG può essere data traducendole nei corrispondenti programmi Prolog, che sono analizzatori sintattici per le grammatiche stesse.

- Nel formalismo delle DCG una grammatica è rappresentata come un insieme di fatti Prolog dove:
  - ogni produzione è un termine Prolog con funtore `-->` (operatore binario infisso)
  - i simboli non terminali sono rappresentati come atomi
  - i simboli terminali sono rappresentati tra parentesi quadre
  - `[]` indica la stringa vuota

DEFINITE CLAUSE GRAMMARS (DCG)

- Vediamo una grammatica che definisce un frammento di linguaggio naturale (lo scopo è il non terminale frase)

```
frase --> fraseNominale, fraseVerbale.
fraseNominale --> articolo, nome, fraseRelativa.
fraseVerbale --> verbo, fraseNominale.
fraseRelativa --> [che], fraseVerbale.
fraseRelativa --> [].
articolo --> [il].
articolo --> [un].
nome --> [cane].
nome --> [gatto].
verbo --> [ama].
verbo --> [mangia].
```

DEFINITE CLAUSE GRAMMARS (DCG)

- Una grammatica espressa mediante DCG può essere facilmente tradotta in un programma Prolog che definisce l'analizzatore della grammatica.
  - Tale programma contiene una clausola per ogni produzione

```
frase --> fraseNominale, fraseVerbale.
```

```
frase(ListIn,ListOut):-
  fraseNominale(ListIn, ListTemp),
  fraseVerbale(ListTemp,ListOut).
```

```
articolo --> [il].
articolo([il|List],List).
```

DEFINITE CLAUSE GRAMMARS (DCG)

- Poiché frase è lo scopo della grammatica, per verificare se una stringa L è una frase generata dalla grammatica, si deve fornire il goal

```
:-frase([il,cane,mangia,un,gatto],[]).
```

- Esistono interessanti estensioni delle DCG per rappresentare grammatiche che definiscono linguaggi più espressivi:
  - uso di parametri/argomenti per i simboli del linguaggio
  - possibilità di associare a una produzione un insieme di predicati Prolog che vengono valutati quando la produzione viene utilizzata

DGC ESTESA

- Esempio:

$G = (V_n, V_t, P, S)$

$V_n = \{E, T\}$

$V_n = \{ +, a \}$

$P = \{ E ::= T + E \mid T$

$T ::= a\}$

$S = E$

Versione DCG

```
e --> t,[+],e.
e --> t.
t --> [a].
```

Versione DCG estesa

```
e(piu(X,Y)) --> t(X),[+],e(Y).
e(X) --> t(X).
t(a) --> [a].
```

DGC ESTESA

- Esempio:

G = (V_n, V_t, P,S)

V_n = {E, T}

V_n = { +, a }

P = { E::= T + E | T

T::=a}

S = E

Versione DCG estesa

```
e(piu(X,Y)) --> t(X),[+],e(Y).
e(X) --> t(X).
t(a) --> [a].
```

Programma Prolog corrispondente

```
e(piu(X,Y),Lin,Lout):-
   t(X,Lin,Lt),piu(Lt,Lt1),
   e(Y,Lt1,Lout).
e(X,Lin,Lout):- t(X,Lin,Lout).
t(a,Lin,Lout):- a(Lin,Lout).
piu([+|L],L).
a([a|L],L).
```

DGC ESTESA

- Esempio:

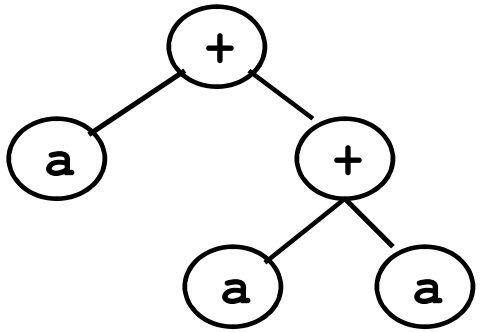
G = (V_n, V_t, P,S)

V_n = {E, T}

V_n = { +, a }

P = { E::= T + E | T

T::=a}



Programma Prolog corrispondente

```
e(piu(X,Y),Lin,Lout):-
   t(X,Lin,Lt),piu(Lt,Lt1),
   e(Y,Lt1,Lout).
e(X,Lin,Lout):- t(X,Lin,Lout).
t(a,Lin,Lout):- a(Lin,Lout).
piu([+|L],L).
a([a|L],L).
```

Il goal

```
:-e(X, [a,+,a,+,a],[]).
yes X = piu(a,piu(a,a)).
STRUTTURA DI UN ALBERO SINTATTICO
ASTRATTO
```

NOTAZIONI INFISSE PREFISSE E POSTFISSE

- Dove posizionare gli operandi rispotto agli operatori ?
  - **prima** → **notazione *prefissa***
    Esempio: ***+ a b***
  - **dopo** → **notazione *postfissa***
    Esempio: ***a b +***
  - **in mezzo** → **notazione *infissa***
    Esempio: ***a + b***

- Le notazioni prefissa e postfissa non hanno problemi di priorità e/o associatività degli operatori
- La notazione infissa richiede regole di priorità e associatività

NOTAZIONI INFISSE PREFISSE E POSTFISSE

- Notazione prefissa:

  `* + 4 5 6`

  - *si legge come (4 + 5) * 6*
  - *denota quindi 54*

- Notazione postfissa:

  `4 5 6 + *`

  - *si legge come 4 * (5 + 6)*
  - *denota quindi 44*

DGC

- Caratteristica importante nelle DCG è che i predicati Prolog possono apparire nella parte destra di una regola
- Quindi, si può tradurre un'espressione appartenente al linguaggio L(G) in notazione polacca postfissa, senza generare l'albero sintattico astratto

$G = (V_n, V_t, P, S)$

$V_n = \{E, T\}$

$V_n = \{ +, a \}$

$P = \{ E ::= T + E \mid T$

$T ::= a\}$

```
e --> t, r.
r --> [+], t, r.
r --> [].
t --> [a].
```

DGC

- Il programma Prolog corrispondente deve contenere alcune chiamate al predicato **`write/1`** per ottenere la stampa della polacca postfissa

```
e --> t, r.
r --> [+], t, r.
r --> [].
t --> [a].
```

- Prima modifica della grammatica

```
e --> t, r.
r --> [+], t, {write(+)}, r.
r --> [].
t --> [a] , {write(a)}.
```

DGC

- Prima modifica della grammatica

```
e --> t, r.
r --> [+], t, {write(+)}, r.
r --> [].
t --> [a] , {write(a)}.
```

- Programma Prolog corrispondente

```
e(Lin,Lout):- t(Lin,Lt), r(Lt,Lout).
r(Lin,Lout):- piu(Lin,Lt), t(Lt,Lt1),
              write(+), r(Lt1,Lout).
r(L,L).
t(Lin,Lout):- a(Lin,Lout), write(a).
piu([+|L],L).
a([a|L],L).
```