6. Parsing top-down

I parser top-down iniziano dalla radice del parse tree e si sviluppano verso le foglie. In questo approccio, viene scelta una produzione e si cerca di farla corrispondere all'input. Se però la scelta fatta si rivela sbagliata, potrebbe essere necessario tornare indietro (**backtracking**).

Nota. Alcune grammatiche, tuttavia, non necessitano di backtracking ma possono essere gestite con il parsing predittivo.

Come detto, un parser top-down inizia dalla radice del parse tree, etichettata con il simbolo obiettivo (**goal**) della grammatica. L'algoritmo di parsing top-down segue questi passaggi:

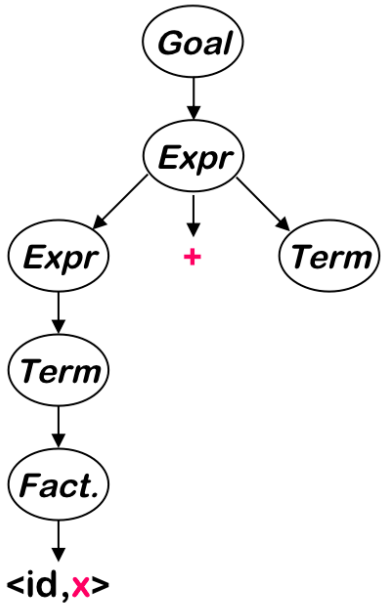
```
Costruisce il nodo radice del parse tree

Ripete finché la parte inferiore dell'albero (detta frangia) non corrisponde
alla stringa di input:

1. In un nodo A etichettato con un simbolo non terminale, seleziona una
produzione che abbia A nel lato sinistro (lhs). Per ogni simbolo nel lato
destro (rhs) della produzione scelta, crea i nodi figli appropriati.

2. Se viene aggiunto un simbolo terminale alla frangia e questo non
corrisponde alla parte dell'input corrispondente, è necessario effettuare
il backtracking

3.Trova il prossimo nodo, etichettato con un simbolo non terminale,
da espandere
```

La chiave del parsing top-down sta nel selezionare la produzione corretta al primo passo. Questa scelta dovrebbe essere guidata dalla stringa di input per minimizzare gli errori e ridurre la necessità di backtracking.

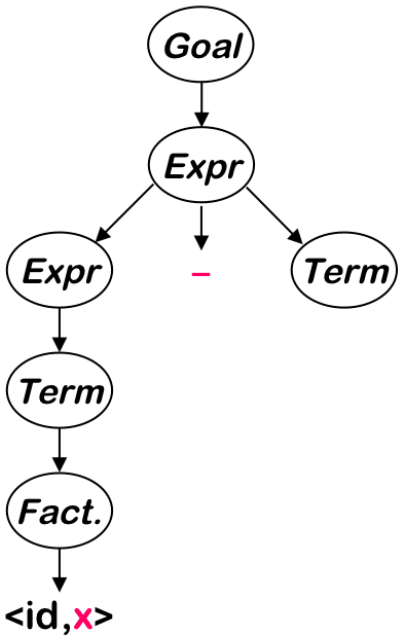
Esempio. Consideriamo la seguente grammatica per le espressioni:

0	*Goal*	→	*Expr*
1	*Expr*	→	*Expr* + *Term*
2		|	*Expr* - *Term*
3		|	*Term*
4	*Term*	→	*Term* * *Factor*
5		|	*Term* / *Factor*
6		|	*Factor*
7	*Factor*	→	(*Expr*)
8		|	number
9		|	id

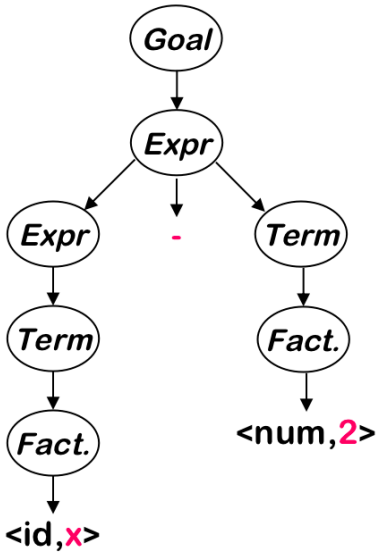
Provando a risolvere l'espressione $x - 2 * y$, possiamo utilizzare un approccio **top-down**. Ecco un esempio di come procedere:



In questo caso, la produzione selezionata si rivela errata, poiché è stato scelto il simbolo "+" invece di "-". Di conseguenza, è necessario tornare indietro (backtracking).



Successivamente, proviamo a espandere il non terminale `Term` con il valore 2



Tuttavia, a questo punto, non è più possibile continuare l'espansione dell'albero di derivazione. Di nuovo, è necessario tornare indietro per correggere la derivazione.

Eliminazione della ricorsione sinistra

La grammatica delle espressioni dell'esempio è ricorsiva a sinistra, e questo può causare la non terminazione in un parser top-down.

Ricordiamo che una grammatica è **ricorsiva a sinistra** quando almeno una delle produzioni di una sua regola ha sul lato destro il simbolo non terminale che appare anche a sinistra

della produzione stessa, e tale simbolo compare come primo elemento. In altre parole, quando un non terminale produce una stringa che inizia con lo stesso non terminale, direttamente o indirettamente

---

Esempio diretto di ricorsività a sinistra: $A \to A\alpha \mid \beta$
Esempio indiretto di ricorsività a sinistra: $A \to B\alpha$, $B \to A\beta$

In un parser top-down, qualsiasi ricorsione deve essere una ricorsione destra per garantire il corretto funzionamento. Pertanto, è necessario convertire la ricorsione sinistra in ricorsione destra.

L'algoritmo sottostante ha l'obiettivo di eliminare la ricorsione sinistra nelle grammatiche e di riformularle in modo che siano adatte per il parsing top-down:

Prima di tutto, i non-terminali (NT) vengono ordinati in una sequenza $A_1, A_2, \ldots, A_n$
Per ogni non-terminale A_i (dove i va da 1 a n):

1. Per ogni A_s precedente (s che va da 1 a $i-1$), ogni volta che troviamo una produzione del tipo $A_i \to A_s\gamma$, cioè una produzione che inizia con A_s, questa viene sostituita:
   - Supponiamo che ci sia una produzione $A_s \to \delta_1 \mid \delta_2 \mid \cdots \mid \delta_k$
   - La regola $A_i \to A_s\gamma$ viene sostituita da:

     $$A_i \to \delta_1\gamma \mid \delta_2\gamma \mid \cdots \mid \delta_k\gamma$$

     Questo passaggio "espande" le produzioni per rimuovere la dipendenza iniziale da A_s.
2. Dopo aver sostituito tutte le produzioni in cui A_i inizia con un A_s precedente, si cerca se A_i ha ricorsione sinistra immediata, cioè una regola del tipo:

   $$A_i \to A_i\alpha$$

   Se viene trovata una produzione di questo tipo, si applica la trasformazione diretta per eliminare la ricorsione sinistra.

Trasformazione diretta. Per eliminare la ricorsione sinistra su un non-terminale A_i, se esiste una produzione della forma $A_i \to A_i\alpha \mid \beta$ (dove α e β non iniziano con A_i) applichiamo la seguente trasformazione:

1. Introduciamo un nuovo non-terminale A_i'.
2. Modifichiamo le produzioni come segue:
   - $A_i \to \beta A_i'$
   - $A_i' \to \alpha A_i' \mid \epsilon$

In questo modo, la nuova grammatica definisce lo stesso linguaggio della vecchia grammatica, utilizzando però solo la ricorsione destra

Scegliere la produzione corretta

La scelta della produzione corretta è fondamentale per il funzionamento efficiente di un parser top-down. Se viene selezionata la produzione sbagliata, il parser deve effettuare il **backtracking**, ossia tornare indietro e riprovare con un'altra produzione.

Un modo per evitare il backtracking è utilizzare il **lookahead**, cioè osservare i simboli successivi nell'input per scegliere la produzione corretta in base al contesto.

Quanto lookahead è necessario?

In generale, potrebbe essere necessario un lookahead arbitrariamente grande per individuare la produzione corretta. Tuttavia, ciò può diventare impraticabile.

Per gestire i casi in cui un lookahead significativo è essenziale, vengono spesso utilizzati algoritmi complessi, come il **Cocke-Younger-Kasami (CYK)** o l'**algoritmo di Earley**.

Fortunatamente, una vasta categoria di grammatiche libere dal contesto può essere analizzata con un lookahead limitato.

Grammatiche LL(1)

In particolare, tra queste classi di grammatiche libere dal contesto, le più interessanti sono le grammatiche **LL(1)** e **LR(1)**. Queste grammatiche possono essere analizzate con un solo simbolo di lookahead. Per ora, ci concentreremo sulle grammatiche **LL(1)** e sul **predictive parsing**, che consentono la selezione della produzione corretta basandosi sul prossimo simbolo dell'input. Questo approccio rende il parser più efficiente e privo di backtracking.

Il termine $LL(1)$ indica alcune caratteristiche specifiche del processo di parsing:

1. **L**: Il primo "L" sta per "left-to-right," cioè l'analisi dell'input viene eseguita da sinistra verso destra.
2. **L**: Il secondo "L" indica che la derivazione avviene **leftmost**, ovvero la derivazione della grammatica procede sempre partendo dall'elemento più a sinistra.
3. **(1)**: Questo numero si riferisce al fatto che il parser necessita di un solo simbolo di **lookahead** per decidere quale produzione grammaticale utilizzare.

Parsing Predittivo

Data una produzione del tipo $A \to \alpha \mid \beta$, il parser deve essere in grado di scegliere l'alternativa corretta tra α e β.

Definizione di FIRST(α)

Per un qualunque α presente nel lato destro delle produzioni di una grammatica G, definiamo l'insieme FIRST(α) come l'insieme di token che possono comparire come primo simbolo in una stringa derivata da α.

In altre parole, un token x appartiene a FIRST(α) se e solo se esiste una derivazione della forma $\alpha \Rightarrow^* x\gamma$, per qualche stringa γ.

Se nella grammatica esistono entrambe le produzioni $A \to \alpha$ e $A \to \beta$, vogliamo che valga la seguente condizione:

$$\mathrm{FIRST}(\alpha) \cap \mathrm{FIRST}(\beta) = \emptyset$$

Questa condizione garantisce che i primi simboli derivabili da α e β siano distinti, permettendo al parser di decidere correttamente quale produzione utilizzando un solo simbolo di lookahead.

Se il parser trova un simbolo che appartiene a FIRST(α), sceglierà la produzione $A \to \alpha$, se invece trova un simbolo che appartiene a FIRST(β), sceglierà la produzione $A \to \beta$.

Produzioni ϵ

Le produzioni ϵ complicano la definizione di una grammatica LL(1) poiché influenzano la capacità del parser di determinare correttamente quale produzione scegliere con un lookahead di un solo simbolo. Se abbiamo due produzioni $A \to \alpha$ e $A \to \beta$ e $\epsilon \in \mathrm{FIRST}(\alpha)$, allora

$$FIRST(\beta) \cap FOLLOW(A) = \emptyset$$

Dove $\mathrm{FOLLOW}(A)$, rappresenta i simboli che possono apparire subito dopo A nella forma sentenziale.

Nota. Una forma sentenziale è una stringa che può essere generata durante il processo di derivazione in una grammatica formale.

Definizione di FIRST$^+(A \to \alpha)$

Per gestire correttamente le produzioni che possono derivare ϵ, definiamo $\mathrm{FIRST}^+(A \to \alpha)$ come segue:

- $\mathrm{FIRST}^+(A \to \alpha) = \mathrm{FIRST}(\alpha) \cup \mathrm{FOLLOW}(A)$, se $\epsilon \in \mathrm{FIRST}(\alpha)$
- $\mathrm{FIRST}^+(A \to \alpha) = \mathrm{FIRST}(\alpha)$, altrimenti

Condizione per una Grammatica LL(1)

Una grammatica è considerata LL(1) se, per ogni coppia di produzioni $A \to \alpha$ e $A \to \beta$, vale la seguente condizione:

$$\mathrm{FIRST}^{+}(A \to \alpha) \cap \mathrm{FIRST}^{+}(A \to \beta) = \emptyset$$

Questa condizione garantisce che il parser possa decidere in modo deterministico quale produzione scegliere in ogni momento, anche in presenza di produzioni ϵ. In altre parole, i simboli che possono essere il primo simbolo derivato da α o β, o che possono seguire immediatamente A nel caso α o β possano derivare ϵ, devono essere disgiunti, così da evitare ambiguità.