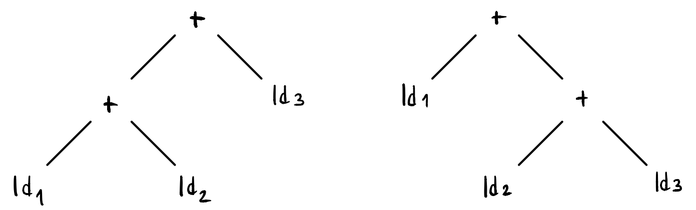
L'eliminazione della ricorsione a sinistra non è una cura rispetto all'ambiguità delle grammatiche.
Per constatare questa affermazione, vediamo un esempio.
La grammatica `E -> E + E | E * E | (E) | id` è ambigua in quanto posso proporre una stringa con due alberi di derivazione con medesima frontiera:
`Id1 + Id2 + Id3`



Eliminazione della ricorsione a sinistra:
$A \rightarrow A\alpha_1 \mid \dots \mid A\alpha_n \mid \beta_1 \mid \dots \mid \beta_k$
Diventa:
$A \rightarrow \beta_1 A' \mid \dots \mid \beta_k A'$
$A' \rightarrow \alpha_1 A' \mid \dots \mid \alpha_n A' \mid \varepsilon$

Nella grammatica precedente: $E \rightarrow E \underbrace{+ E}_{\alpha_1} \mid E \underbrace{* E}_{\alpha_2} \mid \underbrace{(E)}_{\beta_1} \mid \underbrace{id}_{\beta_2}$

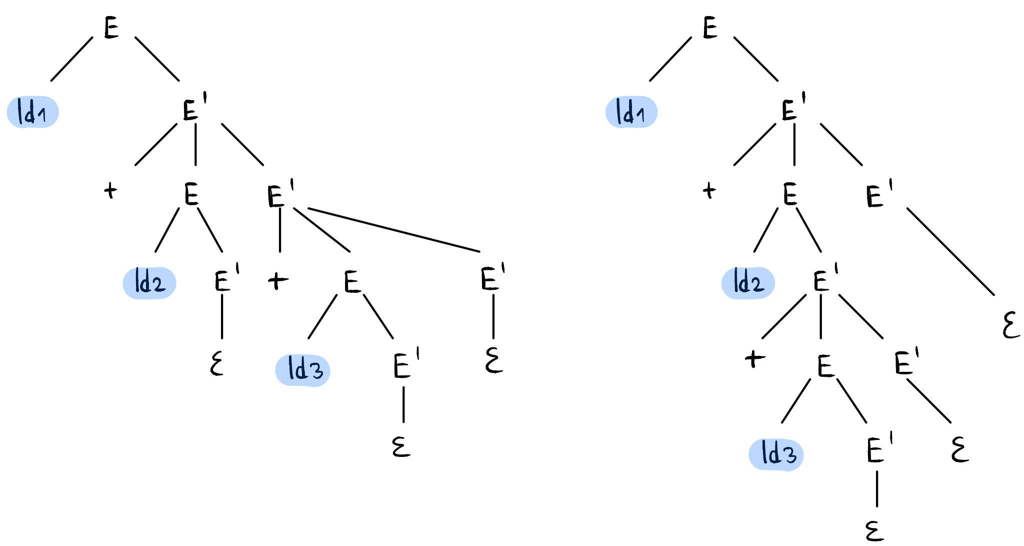
Seguendo le regole specificate poco prima otteniamo:
$E \rightarrow (E)E' \mid id\ E'$
$E' \rightarrow +EE' \mid *EE' \mid \varepsilon$

	FIRST	FOLLOW
E	(, id	$,), +, *
E'	+, *, ε	

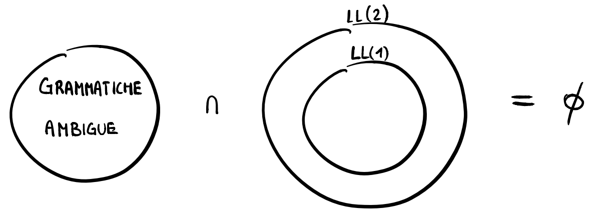
	+	*	id	(	)	$
E			E → id E'	E → (E)E'		
E'	E' → +EE' E → ε	E' → *EE' E → ε			E' → ε	E' → ε

Entry multiply-defined! La grammatica non è LL(1).
Abbiamo constatato che l'eliminazione della ricorsione a sinistra non elimina anche l'ambiguità.

Proviamo ad usare la tabella di parsing per risolvere `Id1 + Id2 + Id3 $`. Otteniamo due alberi distinti ma con medesima frontiera: ambigua!



Attenzione quindi alla seguente relazione insiemistica:



Ora, basta parlare di ricorsione a sinistra; parliamo d'altro.

La grammatica `S -> aSb | ab` è LL(1)? No! Non c'è nemmeno bisogno della tabella. Infatti:

- Ho due produzioni il cui body inizia con lo stesso terminale;
- Per ogni stringa `aSb` e `ab`, i FIRST sono entrambi `a`.

Quindi, nella tabella ho sicuramente la entry

	a
S	$S \to aSb$ $S \to ab$

Questo è il caso di tutte le grammatiche con più di una produzione con gli stessi elementi nei FIRST.

Si parla in questo caso di grammatiche che possono essere **fattorizzate a sinistra**.
La grammatica precedente è un esempio di grammatica che può essere fattorizzata a sinistra, cioè tale che, per qualche A, si hanno produzioni $A \to \alpha_1 \mid \dots \mid \alpha_n$ tali che $\bigcap_{i=1,\dots,n} \text{FIRST}(\alpha_i) \setminus \{\varepsilon\} \neq \emptyset$

Si può riscrivere in una grammatica equivalenti che non presenti questo problema?

Caso generale:
Trasformare $A \to \alpha\beta_1 \mid \alpha\beta_2$ per togliere il prefisso comune α.

$A \to \alpha A'$
$A' \to \beta_1 \mid \beta_2$

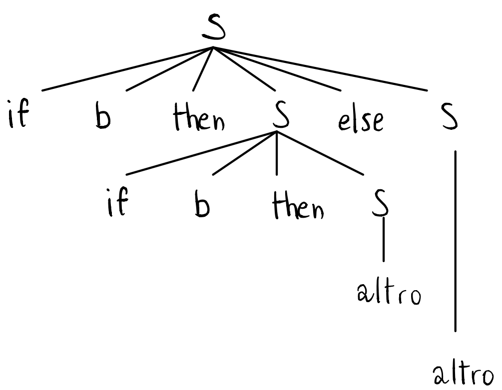
$S \to aSb \mid ab$ Ritardo la scelta sulla produzione
input buffer: a ...

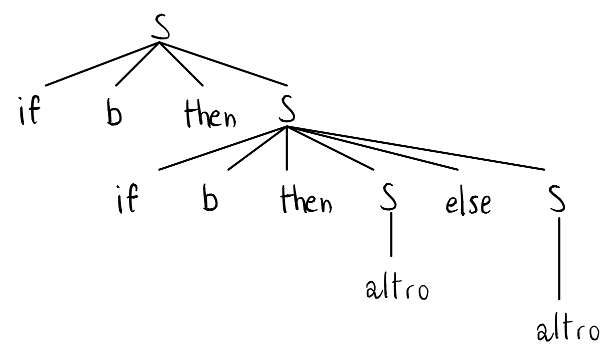
		FIRST	FOLLOW
S	$\to aS'$	a	\$ b
S'	$\to Sb \mid b$	a b	\$ b

	a	b	\$
S	$S \to aS'$		
S'	$S' \to Sb$	$S' \to b$	

Dangling Else

La grammatica seguente, detta del *dangling else*, è ambigua: `S -> if b then S | if b then S else S | altro`
Infatti, nella stringa `if b then if b then altro else altro` non ci si capisce a cosa si accoppi `else` (se al primo o al secondo `if`).





La grammatica è ambigua, su questo non ci piove. Può essere fattorizzata a sinistra?

```
S -> if b then S S'
S' -> else S | ε
```

La nuova versione è LL(1)?

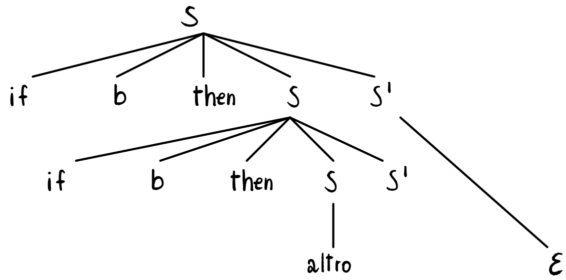
	FIRST	FOLLOW
S	if, altro	\$, else
S'	else, ε	

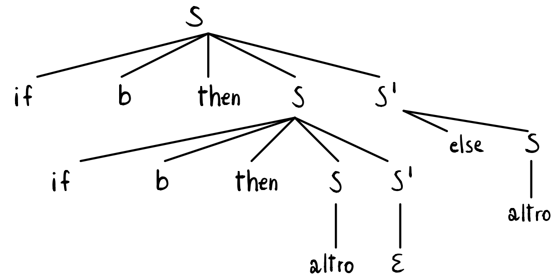
← informazione vuota, ma tecnicamente c'è

	else
S'	$S' \to \text{else } S$ $S' \to \varepsilon$

NON È LL(1).

Oltretutto, è ambigua? Sì, ecco un esempio che lo dimostra:





2 possibilità per disinnescare il dangling else:

- Ogni `then` è abbinato ad 1 `else`;
- Si impone l'**innermost binding**: ogni `else` è abbinato al più vicino `then` unmatched: `if b then if b then altro else altro`

La seconda opzione (innermost binding) si può conseguire facendo uso di una grammatica migliore:
`S -> M | U` (`M` = "Matched", `U` = "Unmatched")
`M - if b then M else M | altro`
`U -> if b then S | if b then M else U`
Questa grammatica consegue l'innermost binding in quanto, tra un `then` e un `else`, ci sono sempre cose `M`atched, cioè non c'è un `then` disaccoppiato.