🟢 Ejemplo 1:

G1:

```less
S → A
A → aA | b
```

🟢 Ejemplo 2:

G2:

```less
S → aS | b
```

🟢 Ejemplo 3:

G3:

```css
S → Aa
A → ε
```

🔴 Ejemplo 4:

G4:

```less
S → Aa | b
A → ε
```

🔴 Ejemplo 5:

G5:

```less
S → aAc | bc | a
A → b
```

🔴 Ejemplo 6:

G6:

```bash
S → L = R | R
L → *R | id
R → L
```

🟢 Ejemplo A1:

```r
S → E
E → E + T | T
T → T * F | F
F → ( E ) | id
```

✅ Típica gramática para expresiones aritméticas sin ambigüedades; es SLR(1).

🟢 Ejemplo A2:

```bash
S → L = R | R
L → id
R → L
```

✅ Aunque autoreferencial, esta gramática puede ser desambiguada mediante FOLLOW sets, es SLR(1).

🟢 Ejemplo A3:

```mathematica
S → if E then S else S | if E then S | a
```

✅ Esta es **ambigua**, pero si se reescribe para eliminar la ambigüedad (ej. usando desambiguación del *else*), puede ser SLR(1).

🔴 Ejemplo B1:

less Copiar Editar

```
S → A a | b
A → ε
```

❌ Conflicto reduce/shift en el FOLLOW(A) con `a` y `b`, como vimos antes.

🔴 Ejemplo B2:

less Copiar Editar

```
S → AaAb | BbBa
A → ε
B → ε
```

❌ Graves conflictos por múltiples caminos de reducción sin distinguir en FOLLOW sets.

🔴 Ejemplo B3:

mathematica Copiar Editar

```
S → id := E | E
E → id
```

❌ Ambigüedad entre `id := id` (asignación) y `id` (expresión sola), sin suficiente información en FOLLOW para deshacer el conflicto.

✅ Ejemplo 1 – Aritmética con paréntesis y prioridad

(Gramática clásica y totalmente LR(1))

```r
E → E + T | T
T → T * F | F
F → ( E ) | id
```

🧠 Esta gramática maneja adecuadamente la precedencia y asociatividad. Aunque **no es LR(0)** ni **SLR(1)** (por conflictos ambiguos en reduce/shift), **sí es LR(1)** al distinguir los lookaheads.

✅ Ejemplo 2 – If-then-else bien desambiguado

(Resuelve el problema del *dangling else*)

```mathematica
S → matched | unmatched
matched → if E then matched else matched | a
unmatched → if E then S | if E then matched else unmatched
```

🧠 Esta gramática es **no LR(0)** ni **SLR(1)**, pero **sí LR(1)**. El uso de lookaheads por producción permite resolver correctamente a qué `if` se asocia cada `else`.

✅ Ejemplo 3 – Listas recursivas con separadores

(Gramática para una lista de IDs separadas por comas)

```bash
S → id L
L → , id L | ε
```

🧠 Esta gramática es **LR(1)** y también puede ser **SLR(1)** dependiendo del análisis. Pero los lookaheads por producción garantizan su parsing correcto.

✅ Ejemplo 4 – Ambigüedad resuelta por contexto

(Asignación vs uso de variable)

```mathematica
S → id := E | E
E → id
```

🧠 Como vimos antes, esto **no es LR(0)** por conflicto shift/reduce, **sí es SLR(1)** y, por supuesto, **sí es LR(1)**. El parser con lookahead sabe cuándo debe esperar un `:=`.

✅ Ejemplo 5 – Recursión indirecta

(Recursión entre no terminales con ε-producción)

```less
S → A
A → B | ε
B → C
C → id
```

🧠 Aunque involucra varios niveles de derivación y un[↓] producción, es una gramática **sin conflictos** si se maneja con lookahead. **Es LR(1)** (y probablemente también SLR(1)).

📌 Ejemplo 2: Conflicto reduce/reduce

less — Copiar — Editar

```
S → AaAb | BbBa
A → ε
B → ε
```

🧠 Casos 2, 4, 6, 10

📌 Ejemplo 3: Clásico shift/reduce con uso de FOLLOW

mathematica — Copiar — Editar

```
S → id := E | E
E → id
```

🧠 Casos 3, 4, 5

📌 Ejemplo 4: Expresiones aritméticas con recursión izquierda

r — Copiar — Editar

```
E → E + T | T
T → T * F | F
F → ( E ) | id
```

🧠 Casos 3, 5, 7

📌 Ejemplo 5: Ambigüedad del "dangling else"

mathematica — Copiar — Editar

```
S → if E then S else S | if E then S | a
```

(reescrita como matched/unmatched)

🧠 Casos 5, 8

📌 Ejemplo 6: Lista recursiva con separador

bash — Copiar — Editar

```
S → id L
L → , id L | ε
```

🧠 Casos 6, 7, 9