3.2: Shift-Reduce-Parser

Es handelt sich bei der gegebenen Grammatik um eine $LR(1)$-Grammatik, d.h. in jedem Schritt wird (höchstens) ein Token *Vorausschau* (*Lookahead*) benötigt, um die nächste Aktion (*shift*, *reduce*) zu bestimmen.

Es soll die folgende Eingabe geparst werden:

```
x = 13 + y * 3;
```

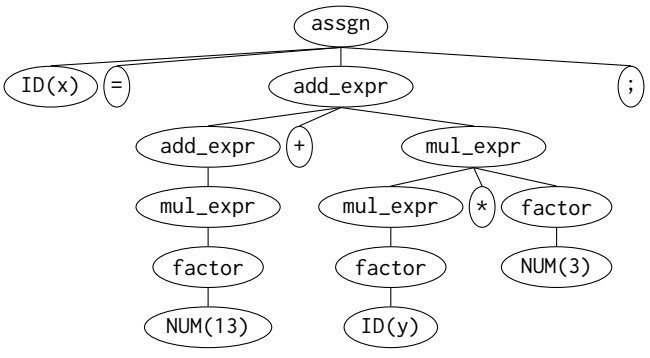
Beim Parsen dieser Eingabe passiert schrittweise Folgendes:

Nr.	Stack	Lookahead	noch nicht gelesen	Aktion
1		`ID(x)`	`= 13 + y * 3;`	Shift
2	`ID(X)`	`=`	`13 + y * 3;`	Shift
3	`ID(X) =`	`NUM(13)`	`+ y * 3;`	Shift
4	`ID(X) = NUM(13)`	`+`	`y * 3;`	Reduce
5	`ID(X) = factor`	`+`	`y * 3;`	Reduce
6	`ID(X) = mul_expr`	`+`	`y * 3;`	Reduce
7	`ID(X) = add_expr`	`+`	`y * 3;`	Shift
8	`ID(X) = add_expr +`	`ID(y)`	`* 3;`	Shift
9	`ID(X) = add_expr + ID(y)`	`*`	`3;`	Reduce
10	`ID(X) = add_expr + factor`	`*`	`3;`	Reduce
11	`ID(X) = add_expr + mul_expr`	`*`	`3;`	Shift
12	`ID(X) = add_expr + mul_expr *`	`NUM(3)`	`;`	Shift
13	`ID(X) = add_expr + mul_expr * NUM(3)`	`;`	*eof*	Reduce
14	`ID(X) = add_expr + mul_expr * factor`	`;`	*eof*	Reduce
15	`ID(X) = add_expr + mul_expr`	`;`	*eof*	Reduce
16	`ID(X) = add_expr`	`;`	*eof*	Shift
17	`ID(X) = add_expr ;`	*eof*		Reduce
18	`assgn`	*eof*		Done

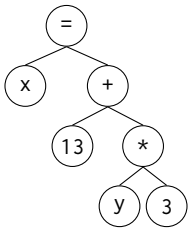
Wichtig:

- In Schritt 2 kann kein *Reduce* zu `factor` ausgeführt werden, da die Eingabe dann nicht mehr geparst werden könnte (erkennbar an dem Lookahead-Zeichen „=“).
- In Schritt 11 kann kein *Reduce* zu `add_expr` ausgeführt werden, da die Eingabe dann nicht mehr geparst werden könnte (erkennbar an dem Lookahead-Zeichen „*)“.

Zugehöriger *konkreter Syntaxbaum*:



Ein geeigneter *abstrakter Syntaxbaum* könnte z.B. wie folgt aussehen:



Insbesondere die Knoten für die „Hilfsproduktionen“ der Grammatik stören und würden die nachfolgenden Analysen und Transformationen auf dem Baum komplizierter machen (Beispiel: die Erkennung faltbarer konstanter Ausdrücke aus Projektübung 2 wäre auf dem konkreten Syntaxbaum deutlich aufwändiger...).