Are there other (possibly better) solutions?

Observation

Although ambiguous, the infix notation is a more intuitive and practical solution!

Elimination of ambiguity of infix notation

- define associativity rules for binary operators
  - addition is left-associative: "`1+1+1`" means "`(1+1)+1`"
  - addition is right-associative: "`1+1+1`" means "`1+(1+1)`"
- define precedence rules for operators, use parentheses to override them
  - multiplication has higher precedence over addition: "`1+1*1`" means "`1+(1*1)`"
  - addition has higher precedence over multiplication: "`1*1+1`" means "`1*(1+1)`"

More on precedence and associativity rules

Operators with the same precedence

- binary operators can have the same precedence; in this case they also share the same associativity rule
  - addition and multiplication have the same precedence and are left-associative: `"1+1*1"` means `"(1+1)*1"` and `"1*1+1"` means `"(1*1)+1"`
  - addition and multiplication have the same precedence and are right-associative: `"1+1*1"` means `"1+(1*1)"` and `"1*1+1"` means `"1*(1+1)"`

Remark on associativity rules

Associativity rules resolve ambiguities between binary operators with the same precedence

Operators with different arities

Mixing together operators of different arities (typically 1, 2 and 3) makes elimination of ambiguity more complex!

Standard techniques for grammar disambiguation

Details

- an ambiguous grammar G is transformed into an *equivalent non-ambiguous* grammar G'
- *equivalent* means that for all non-terminal B of G, the languages generated by G and G' from B are *equal*
- the transformation is able to encode associativity and precedence rules in the non-ambiguous grammar G'

Example 1: $+$ and $\star$ with the same precedence

Ambiguous grammar

```
Exp ::= Num | Exp '+' Exp | Exp '*' Exp | '(' Exp ')'
Num ::= '0' | '1'
```

Non-ambiguous grammar, left associative operations

```
Exp ::= Atom | Exp '+' Atom | Exp '*' Atom
Atom ::= Num | '(' Exp ')'
Num ::= '0' | '1'
```

Non-ambiguous grammar, right associative operations

```
Exp ::= Atom | Atom '+' Exp | Atom '*' Exp
Atom ::= Num | '(' Exp ')'
Num ::= '0' | '1'
```

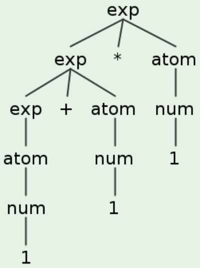
Example 1: $+$ and $\star$ with the same precedence

Non-ambiguous grammar, left associative operations

```
Exp ::= Atom | Exp '+' Atom | Exp '*' Atom
Atom ::= Num | '(' Exp ')'
Num ::= '0' | '1'
```

Remark: `Exp '+'Atom` (`Exp '*'Atom`) means that on the right side of $+$ ($\star$) additions (multiplications) are allowed only if surrounded by parentheses

Unique derivation tree for `1+1*1`



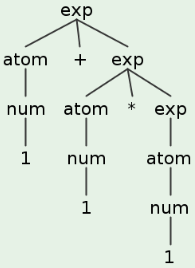
Example 1: $+$ and $\star$ with the same precedence

Non-ambiguous grammar, right associative operations

```
Exp ::= Atom | Atom '+' Exp | Atom '*' Exp
Atom ::= Num | '(' Exp ')'
Num ::= '0' | '1'
```

Remark: `Atom '+'Exp` (`Atom '*'Exp`) means that on the left side of $+$ ($\star$) additions (multiplications) are allowed only if surrounded by parentheses

Unique derivation tree for `1+1*1`



Example 2: $\star$ with higher precedence

Ambiguous grammar

```
Exp ::= Num | Exp '+' Exp | Exp '*' Exp | '(' Exp ')'
Num ::= '0' | '1'
```

Non-ambiguous grammar, left associative operations

```
Exp ::= Mul | Exp '+' Mul
Mul ::= Atom | Mul '*' Atom
Atom ::= Num | '(' Exp ')'
Num ::= '0' | '1'
```

Non-ambiguous grammar, right associative operations

```
Exp ::= Mul | Mul '+' Exp
Mul ::= Atom | Atom '*' Mul
Atom ::= Num | '(' Exp ')'
Num ::= '0' | '1'
```

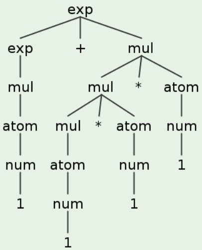
Example 2: $\star$ with higher precedence

Non-ambiguous grammar, left associative operations

```
Exp ::= Mul | Exp '+' Mul
Mul ::= Atom | Mul '*' Atom
Atom ::= Num | '(' Exp ')'
Num ::= '0' | '1'
```

Remark: `Mul '*'Atom` means that on both sides of $\star$ additions are allowed only if surrounded by parentheses

Unique derivation tree for `1+1*1*1`



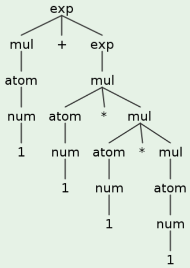
Example 2: $\star$ with higher precedence

Non-ambiguous grammar, right associative operations

```
Exp ::= Mul | Mul '+' Exp
Mul ::= Atom | Atom '*' Mul
Atom ::= Num | '(' Exp ')'
Num ::= '0' | '1'
```

Remark: `Atom '*'Mul` means that on both sides of $\star$ additions are allowed only if surrounded by parentheses

Unique derivation tree for `1+1*1*1`



Remaining examples

Solutions left for exercise

- $\star$ higher precedence and left associative, $+$ right associative
- $\star$ higher precedence and right associative, $+$ left associative
- $+$ higher precedence and left associative, $\star$ right associative
- $+$ higher precedence and right associative, $\star$ left associative

The non-ambiguous grammars can be easily defined by symmetry

Ambiguous syntax for statements

Ambiguity: not only expressions ...

```
Stmt ::= ID '=' Exp | 'if' '(' Exp ')' Stmt | Stmt ';' Stmt
  | '{' Stmt '}'
Exp ::= ID | BOOL // ID and BOOL defined by regular expressions
```

Two derivation trees for "if(x)x=false;y=true"