木構造と数式処理

第３回　数式の変形

- 再帰アルゴリズム
- アルゴリズムの設計

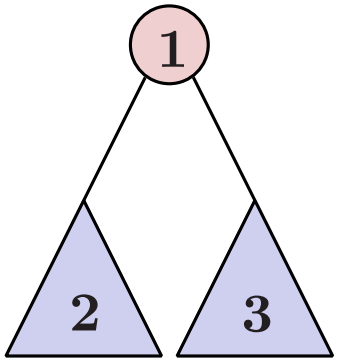
山田 俊行

http://www.cs.info.mie-u.ac.jp/~toshi/

２分木上の再帰の基本

先行順

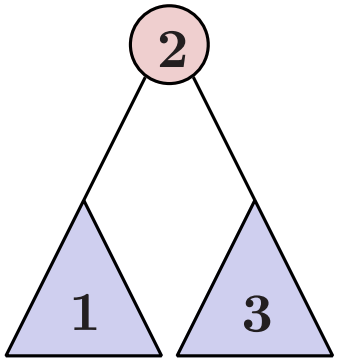
preorder



根 → 左の子 → 右の子

中間順

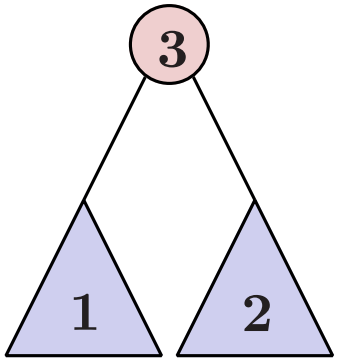
inorder



左の子 → 根 → 右の子

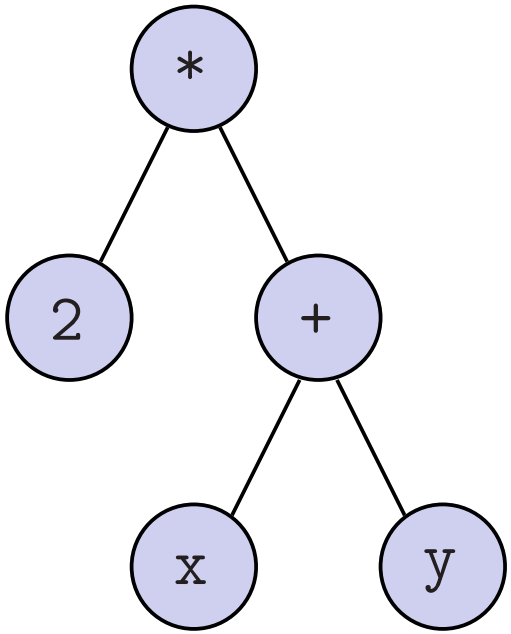
後行順

postorder



左の子 → 右の子 → 根

２分木上の再帰の例

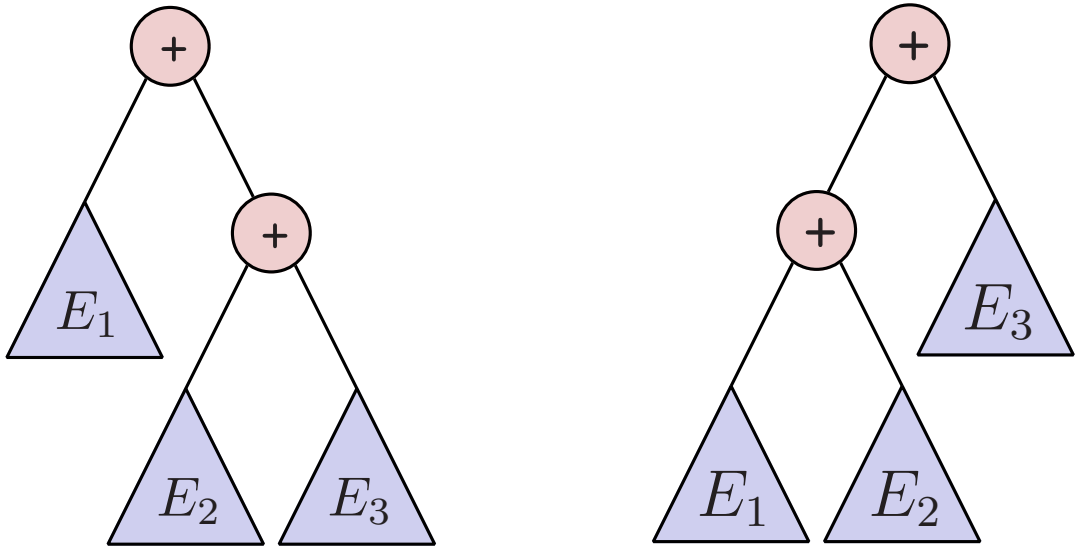


`show_tree()`	$\rightarrow$	`*(2(#,#),+(x(#,#),y(#,#)))`	先行順
`show_exp()`	$\rightarrow$	`(2*(x+y))`	中間順
`eval_exp()`	$\rightarrow$	`4`	後行順

再帰アルゴリズム

左結合（括弧の左くくり）の基本操作

$$(E_1 + (E_2 + E_3)) \quad \rightsquigarrow \quad ((E_1 + E_2) + E_3)$$

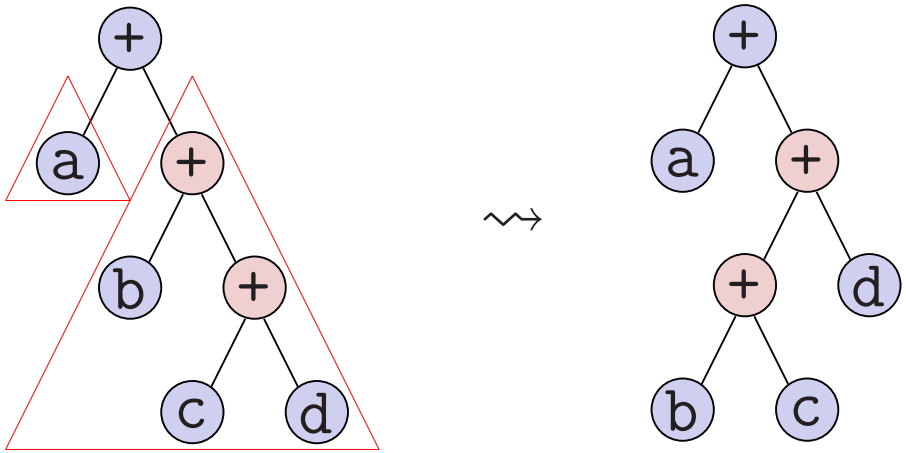


$\Rightarrow$ 式全体にわたる左結合は，再帰的な左結合で実現

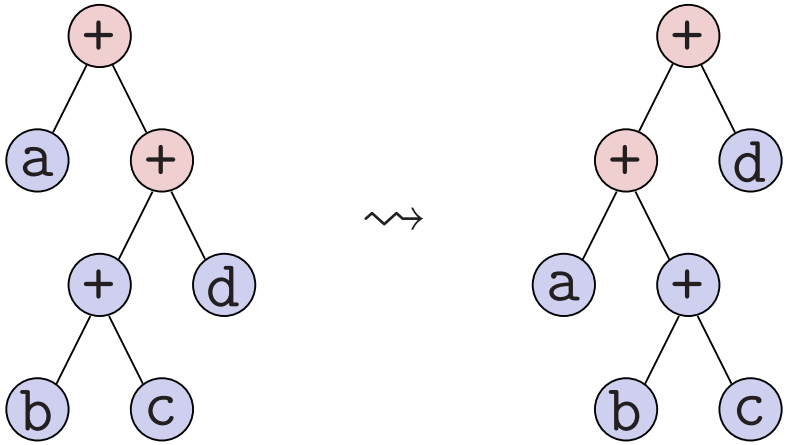
```
(a+(b+(c+d)))   ⇝   (((a+b)+c)+d)
```

再帰アルゴリズムの例

1. 部分式の再帰的な左結合



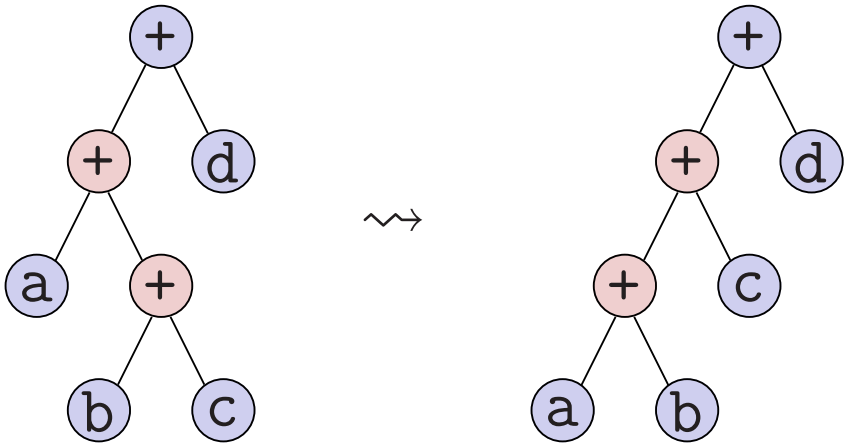
2. 式全体の１段階の左結合（必要なら）



再帰アルゴリズムの例（続き）

3. 再び，再帰的な左結合

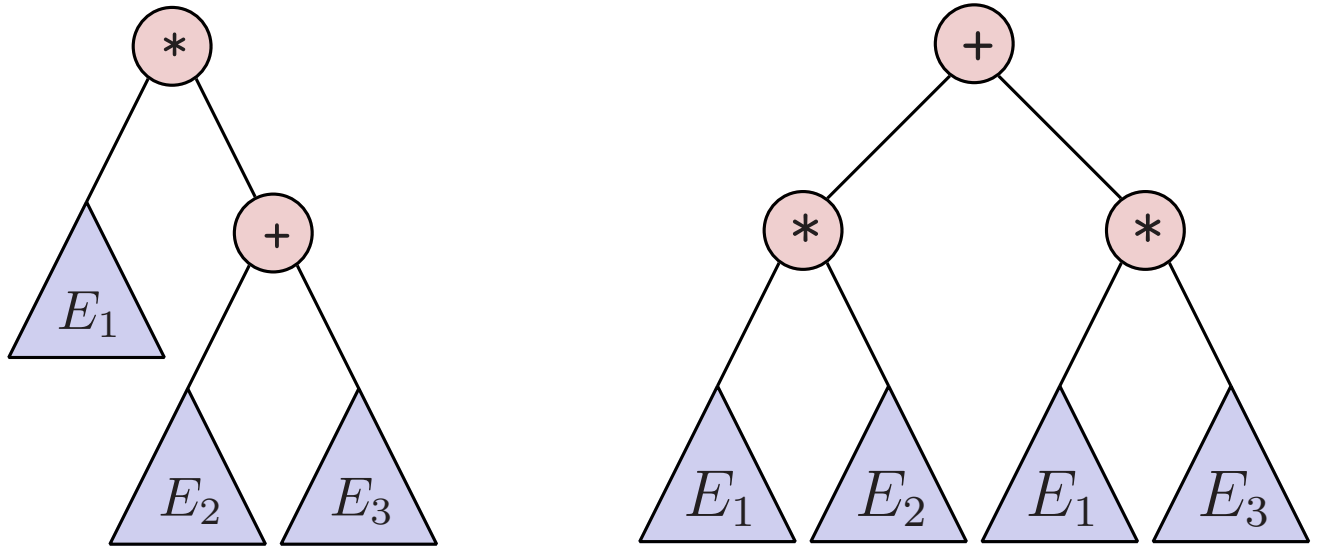
変形できる部分式が生じる可能性を考慮



アルゴリズムの設計

積の和に対する分配

$$(E_1 \times (E_2 + E_3)) \quad \rightsquigarrow \quad ((E_1 \times E_2) + (E_1 \times E_3))$$



式全体にわたる分配

$\Rightarrow$ 基本操作 と その適用手順 を定める