木構造と数式処理

第１回　２分木のデータ構造

- 数式処理
- 木
- ２分木のデータ構造
- 再帰手続き

山田 俊行

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

数式処理

- 数式に基づいて数値を計算
- 数式を操作

$\Rightarrow$ 数値計算の他に，文字を含む式の操作ができる

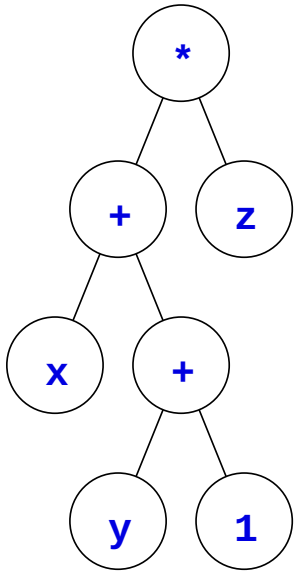
数式処理システム

ウェブ上の資料「数式処理システム」に沿って，
Maxima の基本機能を紹介

木

- 構造をもつデータ（数式など）の基本的な表し方
- 動的な管理に向く，階層的な構成のデータ構造

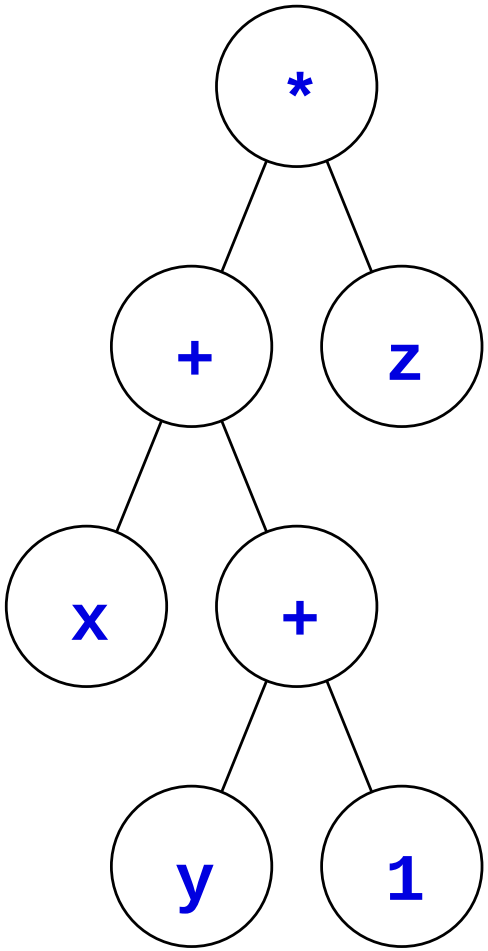
数式の木表現

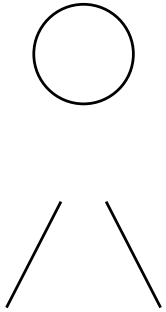


$(x + (y + 1)) \times z$ を表す木

$\Rightarrow$ 部分式が部分木に対応

木の基本用語

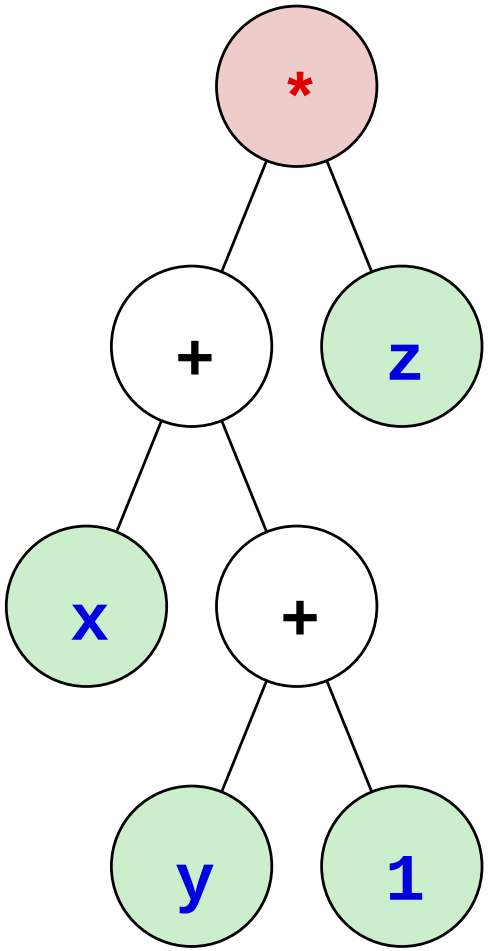


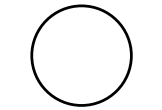


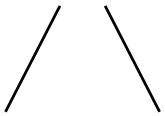
… 節 (node)

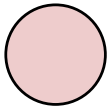
… 枝 (branch)

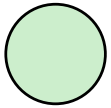
木の基本用語

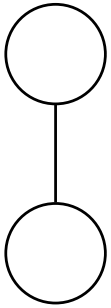



… 節 (node)


… 枝 (branch)

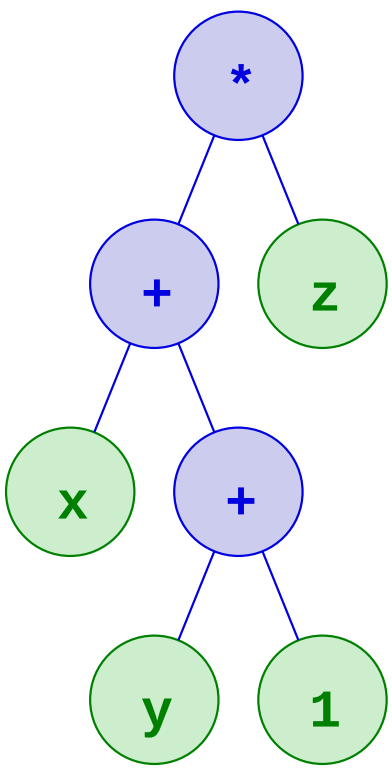

… 根 (root)

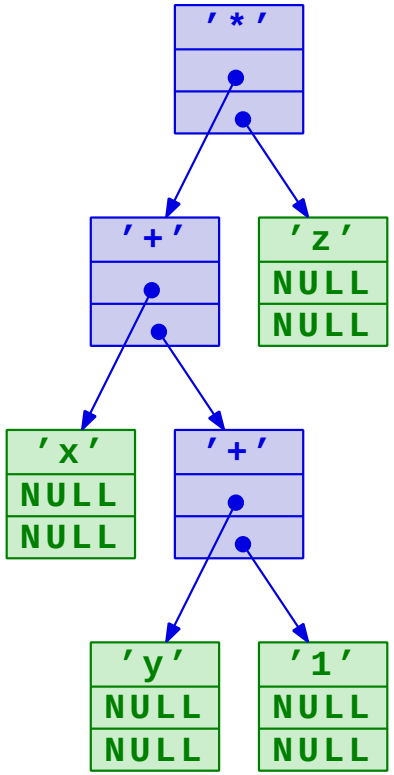

… 葉 (leaf)


… 親 (parent)

… 子 (child)

２分木のデータ構造





動的に節を生成する２種類の手続きを用意

- 枝のある節 の生成
- 枝のない節 (葉) の生成

再帰手続き (関数)

- 引き数を変えて，自分自身を呼び出す手続き (関数)
- 再帰的なデータ構造 (木構造など) の処理を簡潔に書ける

再帰手続きの例

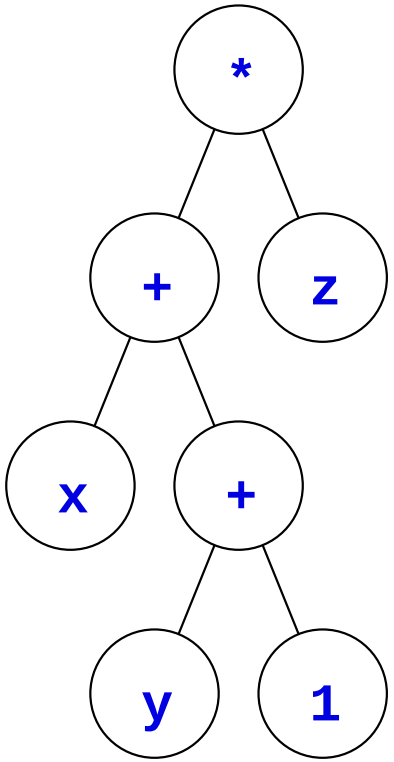
```
printNodes(Ｔ) {
  if (Ｔが葉){ print(Ｔの根のデータ); }
  else       { print(Ｔの根のデータ);
               printNodes(Ｔの左部分木);
               printNodes(Ｔの右部分木); }
}
```

根 → 左部分木 → 右部分木 の順の再帰的な表示

再帰手続きの実行例

根 → 左部分木 → 右部分木 の順の再帰的な表示

```
    printNodes( (x+(y+1))*z )
*
      printNodes( x+(y+1) )
+
        printNodes( x )
x
        printNodes( y+1 )
+
          printNodes( y )
y
          printNodes( 1 )
1
      printNodes( z )
z
```



⇒ 深さ優先探索で節を初めて訪れるときの表示と同じ

再帰関数の例

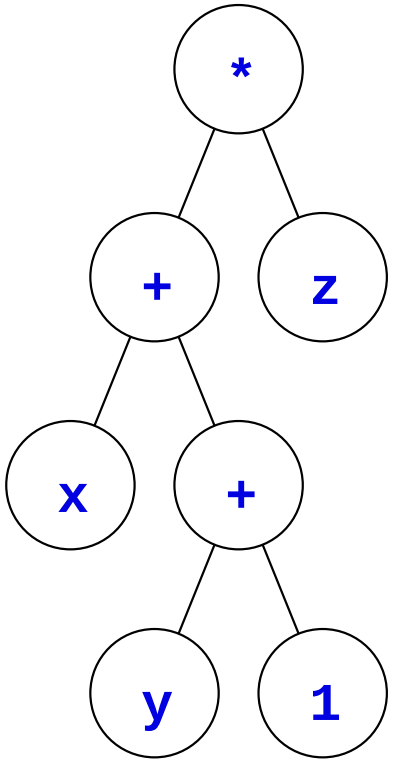
```
numNodes(Ｔ) {
  if (Ｔが葉)return  1;
  else       return (1 + numNodes(Ｔの左部分木)
                       + numNodes(Ｔの右部分木));
}
```

- 葉だけの木の節数 $=$ 1
- 枝のある木の節数 $=$ $1 +$ 左部分木の節数 $+$ 右部分木の節数

再帰関数の実行例

木の節数の計算

```
numNodes( (x+(y+1))*z )
  numNodes( x+(y+1) )
    numNodes( x )
    1                          x の節数
    numNodes( y+1 )
      numNodes( y )
      1                        y の節数
      numNodes( 1 )
      1                        1 の節数
    3                        y+1 の節数
  5                        x+(y+1) の節数
  numNodes( z )
  1                          z の節数
7                        (x+(y+1))*z の節数
```



⇒ 深さ優先探索で節の訪問を終えると，部分木の値が決まる