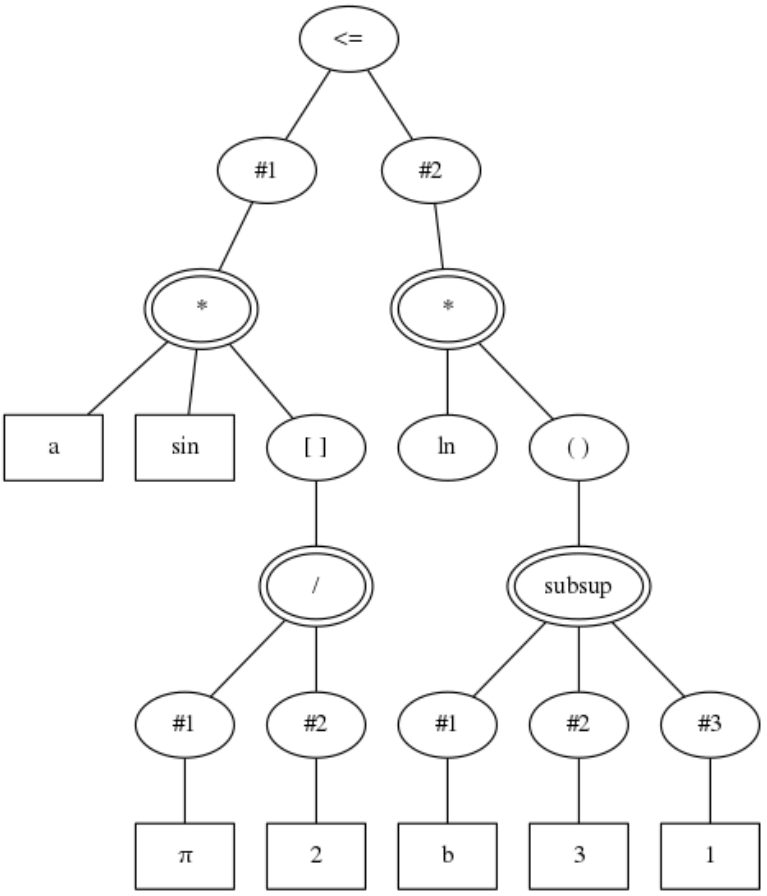
公式

$$a \cdot \sin\left[\frac{\pi}{2}\right] = \ln(b_1^3)$$

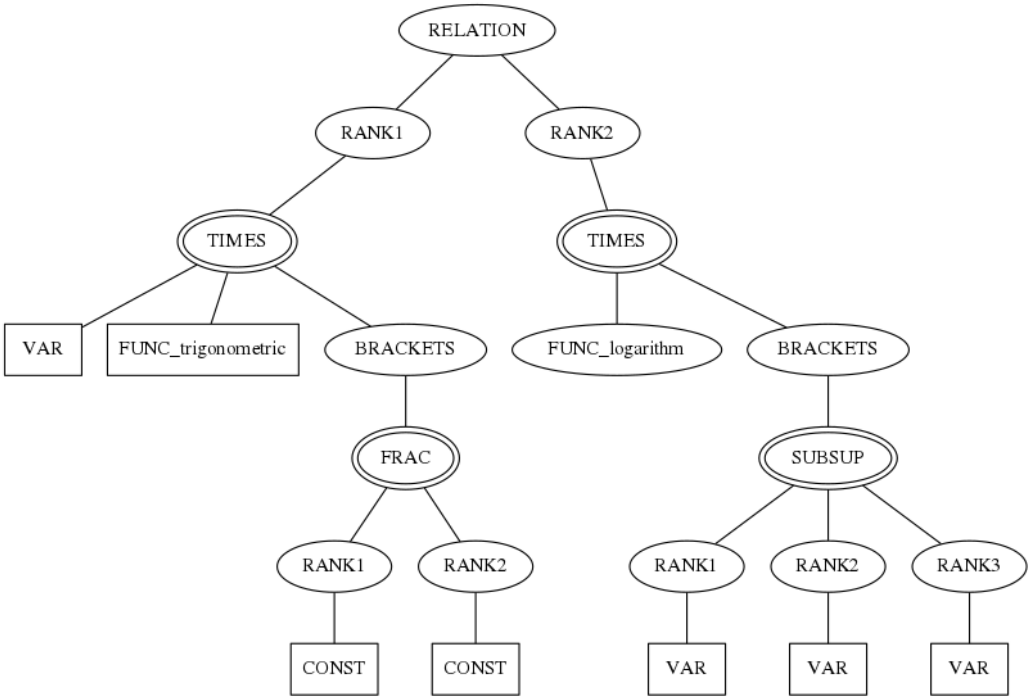
在 parse 以后的 operator tree:



其中：

1. commutative operator 需要在儿子节点上添加 pseudo node (e.g. #1, #2 ...)，以在 leaf-root path 中区分儿子的顺序
2. 图中双边的节点叫做“gener node”，之后会对他们进行一些冗余 index，以此来提供对任意（或多个）gener node(s) 替换成 wildcard variable(s) 之后的 wildcard 搜索功能

上图 operator tree 是按照各个节点所对应的"symbol"来绘制的。其实每个"symbol"经过 tokenization 后可以得到所谓的"token"，如果用 token 来表示这棵树：



在我们的搜索引擎中，token 用来扩大搜索范围，而 symbol 用来在这个范围内给搜索结果打分。比如对于 query: $a \cdot \ln(b)$，表达式 $\lambda \cdot \ln(b)$ 和 $x \times \log(y)$ 都可以被搜索到（他们的 token 表示都相同），但是前者的排位在搜索结果中会更高(因为前者的 symbol 更多地符合 query)。

从这个 operator tree 上，我们提取 leaf-root paths (lr_path):

```
struct lr_path_head {
    list lr_path;
    lf_symbol_id; /* leaf node symbol ID */
    lf_id;        /* leaf node ID */
};

struct lr_path_ele {
    token_id;
    rank;     /* the rank among brothers */
    gener;    /* boolean: is it gener node? */
    sons;     /* number of sons */
};
```

其中，gener node 的充分条件是：

1. 是内节点（不是 root 也不是 leaf)
2. 儿子数目大于一

根据 leaf-root path，我们按 path 中的 token 顺序定位 posting list 所在的目录路径。比如图中最左边的那条 leaf-root path 将会在 index 的时候定位到 `${index_dir}/VAR/TIMES/RANK1/RELATION`，并把这条 path 的 posting_item 和 lr_path_info_item 结构 append 到（经过压缩）posting list 二进制文件 `${index_dir}/VAR/TIMES/RANK1/RELATION/{posting.bin, pathinfo.bin}`.

压缩前的 posting_item 和 lr_path_info_item 数据结构：

```
struct posting_item {
    doc_id:                 32; /* delta and bit pack compression */
    frml_id:                32; /* bit pack compression */
    lr_path_info_pos:       32; /* delta and bit pack compression */
};

struct lr_path_info_item {
    lf_symbol_id: 8;  /* bit pack compression */
    br_hash:      7;  /* leaf-root branch (excluding leaf node) symbols' HASH value */
    eoi:          1;  /* boolean: end of leaf-root path info item? */
};
```