```
var traverseDepthFirstWithPrefix = function (prefix, callback, nodelist) {
    (function recurse(currentNode) {
        prefix(currentNode);
        for (var i = 0, length = currentNode.children.length; i < length; i++) {
            recurse(nodelist[currentNode.children[i]]);
        }
        callback(currentNode);
    })(nodelist[0]);
};
var traverseDepthFirst = function (callback, nodelist) {
    (function recurse(currentNode) {
        for (var i = 0, length = currentNode.children.length; i < length; i++) {
            recurse(nodelist[currentNode.children[i]]);
        }
        callback(currentNode);
    })(nodelist[0]);
};
var traverseRootFirst = function (callback, nodelist) {
    (function recurse(currentNode) {
        callback(currentNode);
        for (var i = 0, length = currentNode.children.length; i < length; i++) {
            recurse(nodelist[currentNode.children[i]]);
        }
    })(nodelist[0]);
};
var traverseSimple = function (callback, nodelist) {
    for (var i = 0; i < nodelist.length; i++) {
        var node = nodelist[i];
        callback(node);
    }
};
function parse_brackets(tree) {
    var list_of_nodes = tree.nodelist;
    var index_of_last_node = 0;
    var index = 0;
    var stop = false;
    do {
        var node = list_of_nodes[index];
        //console.log(index + ': ' + node.content);
        var left_pos = node.addBracket(tree);
        index_of_last_node = tree.nodelist.length;
        //console.log('left_pos=' + left_pos + ' index_of_last_node =' + index_of_last_node);
        if (left_pos === -1) {
            index++;
            if (index >= index_of_last_node) {
                stop = true;
            }
        }
    } while (stop === false);
    return list_of_nodes;
}

function parse_frac(tree) {
```

```
    var list_of_nodes = tree.nodelist;
    for (var i = 0; i < list_of_nodes.length; i++) {
        var node = list_of_nodes[i];
        //if (node.type.startsWith('unknown')) {
        if (node.content === '\\frac§§' && node.children.length === 2) {
            console.log('found \\\\frac§§ at ' + node.id);
            node.type = 'frac';
            node.content = 'frac';
        }
        //}
    }
}
;
function function_list() {
    var result = [];
    result.push('sinh');
    result.push('cosh');
    result.push('tanh');
    result.push('sin');
    result.push('cos');
    result.push('tan');
    result.push('ln');
    result.push('log');
    result.push('exp');
    return result;
}
;

function parse_function(tree) {
    var i = 0;
    // length of tree.nodelist may change ->
    // do not use "for", but "do-while"
    do {
        var node = tree.nodelist[i];
        var stop_fu = false;
        var k = 0;
        var pos = -1;
        do {
            var fu = function_list()[k];
            var type = 'fu-' + fu;
            fu = '\\' + fu;
            // console.log('searching for ' + fu);
            pos = node.content.indexOf(fu);
            if (pos > -1) {
                var pow = '';
                var leftpart = node.content.substring(0, pos);
                var left_count = (leftpart.match(/§/g) || []).length;
                var rest = node.content.substring(pos + fu.length);
                var fu_node = create_node(type, '', tree);
                // link node <-> fu_node
                fu_node.parent = node.id;
                console.log('left_count=' + left_count + 'id=' + node.id + ' children=' + node.children);
                var remember = node.children[left_count] || 0;
```

```
//                  console.log('remember1=' + remember);
            node.children[left_count] = fu_node.id;
//                  console.log('remember2=' + remember);
            if (rest.startsWith('^')) {
                //fu-power
                fu_node.content = 'power';
                rest = rest.substring(1);
                console.log('found ' + fu + '^ at ' + node.id + ' rest=' + rest);
                if (rest.startsWith('§')) {
                    // \sin^§...
                    pow = '§';
                    rest = rest.substring(1);
                    if (rest.startsWith('§')) {
                        // \sin^§§...
                        fu_node.children[0] = remember;
                        tree.nodelist[remember].parent = fu_node.id;
                        fu_node.children[1] = node.children[left_count + 1];
                        tree.nodelist[node.children[left_count + 1]].parent = fu_node.id;
                    } else {
                        // \sin^§x
                        var arg = create_node('unknown leaf', rest, tree);
                        fu_node.children[0] = remember;
                        tree.nodelist[remember].parent = fu_node.id;
                        fu_node.children[1] = arg.id;
                        arg.parent = fu_node.id;
                    }
                } else {
                    // \sin^3...
                    pow = rest.substr(0, 1);
                    rest = rest.substring(1);
                    if (rest.startsWith('§')) {
                        // \sin^3§
                        var node_pow = create_node('unknown leaf', pow, tree);
                        fu_node.children[0] = node_pow.id;
                        node_pow.parent = fu_node.id;
                        fu_node.children[1] = remember;
                        arg.parent = fu_node.id;
                    } else {
                        // \sin^32\alpha
                        var node_pow = create_node('unknown leaf', pow, tree);
                        var arg = create_node('unknown leaf', rest, tree);
                        fu_node.children = [node_pow.id, arg.id];
                        node_pow.parent = fu_node.id;
                        arg.parent = fu_node.id;
//                                  console.log('remember3=' + remember);
                        tree.nodelist[remember].parent = fu_node.id;
                    }
                }
                console.log('type=' + type + ' pow=' + pow + ' rest=' + rest);
            } else {
                // no power: \sin...
                if (rest.startsWith('§')) {
                    // \sin§
```

```
                fu_node.children[0] = remember;
                tree.nodelist[remember].parent = fu_node.id;
            } else {
                // \sin2\alpha
                var arg = create_node('unknown leaf', rest, tree);
                fu_node.children[0] = arg.id;
                arg.parent = fu_node.id;
            }
            console.log('found ' + fu + ' at ' + node.id + ' rest=' + rest);
        }
        node.content = leftpart + '§';
    }
    k++;
    if (k > function_list().length) {
        stop_fu = true;
    }
  } while (stop_fu === false);
  i++;
} while (i < tree.nodelist.length);
}
;

function parse_int(tree) {
  // for (var i = 0; i < list_of_nodes.length; i++) {
  // does not fit because length of list changes
  var i = 0;
  var stop = false;
  do {
    var node = tree.nodelist[i];
    var content = node.content;
    if (content.startsWith('\\int')) {
      console.log('****** \\int found at node # ' + node.id);
      var pos_sub = content.indexOf('_');
      var pos_pow = content.indexOf('^');
      var rest = '';
      console.log('sub found at ' + pos_sub + ' pow found at ' + pos_pow);
      if (pos_sub === -1 || pos_pow === -1) {
        // indefinite integral
        rest = content.substring(4); //remove \\int = 4 chars
        node.type = 'indefinite_integral';
        console.log(node.type + ' ' + rest);
      } else {
        // definite integral
        var lower_bound = content.substring(pos_sub + 1, pos_pow);
        var upper_bound = content.substring(pos_pow + 1, pos_pow + 2);
        rest = content.substring(pos_pow + 2);
        console.log(lower_bound + ' ' + upper_bound + ' rest=' + rest);
        // for every *_bound which is no § an unshift of children[] is necessary
        // check # of §
        var lower_count = (lower_bound.match(/§/g) || []).length;
        if (lower_count === 0) {
          // no bracket, new node needed
          var lower = create_node('lower_bound', lower_bound, tree);
```

```
            lower.parent = node.id;
            //console.log('before unshift: node,children.length=' + node.children.length);
            node.children.unshift(lower.id);
            //console.log('after unshift: node,children.length=' + node.children.length);
            // now children[0] is free
            // node.children[0] = lower.id;
          } else {
            // children[0] stays at ist place and contains id of bracket
          }
          // console.log('children=' + node.children);
          var upper_count = (upper_bound.match(/$/g) || []).length;
          if (upper_count === 0) {
            var upper = create_node('upper_bound', upper_bound, tree);
            upper.parent = node.id;
            node.children.unshift(0); //dummy
            // now children[0] is free, but we need children[1]
            node.children[0] = node.children[1];
            node.children[1] = upper.id;
          } // else do nothing
          console.log('children=' + node.children);
          // check # of brackets ($)
          var rest_count = (rest.match(/$/g) || []).length;
          if (node.children.length !== 2 + rest_count) {
            throw('(parse_int) Wrong number of bracket markers');
          }

          node.type = 'definite_integral';
          node.content = rest;
          console.log(node.type);
          console.log('lower_bound=' + node.children[0] + ' upper_bound=' + node.children[1] + '
          rest=' + node.content);
        }
      }
      // console.log('i=' + i + ' tree.nodelist.length=' + tree.nodelist.length + 'content=' +
      node.content);
      i++;
      if (i === tree.nodelist.length) {
        stop = true;
      }
    } while (stop === false);
}
;
function parse_sqrt(tree) {
  parse_radix(tree, false);
}
;
function parse_nthroot(tree) {
  parse_radix(tree, true);
}
;
function parse_radix(tree, nthroot) {
  var i = 0;
  var stop = false;
```

```
var needle = '\\sqrt$';
if (nthroot === true) {
    needle += '$';
}
do {
    var node = tree.nodelist[i];
    var pos = -1;
    if (node.type.startsWith('unknown')) {
        do {
            pos = node.content.indexOf(needle);
            if (pos > -1) {
                var left = node.content.substring(0, pos);
                var right = node.content.substring(pos + needle.length);
                var rad_index = (left.match(/$/g) || []).length;
                // if there is no $ in left, then rad_index = 0
                console.log(i + ' content=' + node.content + ' pos=' + pos);
                console.log(' left=###' + left + '###' + ' right=###' + right + '###');
                if (nthroot === true) {
                    var newcontent = left + '$' + right;
                    // node has one $ less!
                    console.log('new=' + newcontent);
                    node.content = newcontent;
                    //check
                    var test = tree.nodelist[node.children[rad_index]].type;
                    console.log(test + ' should be bracket-[');
                    test = tree.nodelist[node.children[rad_index + 1]].type;
                    console.log(test + ' should be bracket-{');
                    var radix = create_node('nthroot', '', tree);
                    // link radix
                    radix.parent = node.id;
                    //radix has two children
                    radix.children = [node.children[rad_index], node.children[rad_index + 1]];
                    // now the other directions
                    tree.nodelist[node.children[rad_index]].parent = radix.id;
                    tree.nodelist[node.children[rad_index + 1]].parent = radix.id;
                    node.children[rad_index] = radix.id;
                    node.children.splice(rad_index + 1, 1);
                } else {
                    var newcontent = left + '$' + right;
                    console.log('new=' + newcontent);
                    //check
                    var test = tree.nodelist[node.children[rad_index]].type;
                    console.log(test + ' should be bracket-{');
                    node.content = newcontent;
                    var radix = create_node('sqrt', '', tree);
                    // link radix
                    radix.parent = node.id;
                    //radix has only one child
                    radix.children = [node.children[rad_index]];
                    // now the other directions
                    tree.nodelist[node.children[rad_index]].parent = radix.id;
                    node.children[rad_index] = radix.id;
                }
```

```
            }
        } while (pos > -1);
    }
    i++;
    if (i === tree.nodelist.length) {
        stop = true;
    }
  } while (stop === false);
}

function parse(tree) {
  result = parse_brackets(tree);
  console.clear();
  console.log('brackets');
//    traverseSimple(
//            function (node) {
//                node.debug(tree.nodelist);
//            }, tree.nodelist);
  console.log('plusminus');
  result = remove_operators(tree, 'plusminus');
//    traverseSimple(
//            function (node) {
//                node.debug(tree.nodelist);
//            }, tree.nodelist);
  console.log('timesdivided');
  result = remove_operators(tree, 'timesdivided');
//    traverseSimple(
//            function (node) {
//                node.debug(tree.nodelist);
//            }, result);
//
  console.log('integral');
  parse_int(tree);
  console.log('square root / nth root');
  parse_nthroot(tree);
  parse_sqrt(tree);
  //traverseDepthFirst(
  traverseSimple(
        function (node) {
            node.debug(tree.nodelist);
        }, tree.nodelist);
  console.log('function');
  parse_function(tree);
  console.log('power');
  result = remove_operators(tree, 'power');
//    traverseSimple(
//            function (node) {
//                node.debug(tree.nodelist);
//            }, tree.nodelist);
  console.log('sub');
  result = remove_operators(tree, 'sub');
  var list_of_free = delete_single_nodes(tree);
  console.log('frac');
```

```
  parse_frac(tree);
  traverseSimple(
      function (node) {
        node.debug(tree.nodelist);
      }, tree.nodelist);
}
;
function tree2TEX(tree) {
  var depth = 0;
  return recurse(tree.root);
  function recurse(node) {
    var number_of_childs = (node.children || []).length;
    // console.log('children=' + node.children);
    depth++;
    var res = [];
    for (var i = 0; i < number_of_childs; i++) {
      var child = tree.nodelist[node.children[i]];
      res[i] = recurse(child);
    }
    var done = false;
    if (number_of_childs === 0) {
      // leaf
      result = node.content;
    }
    if (number_of_childs === 1) {
      if (node.type.startsWith('root')) {
        result = res[0];
        done = true;
      }
      if (node.type.startsWith('bracket')) {
        result = node.type.substring(8);
        var pos = ['(','[','{','\\left(','\\left[','\\left\\{'].indexOf(result);
        if (pos === -1) {
          var rightbra = 'no corresponding bracket found error';
        } else {
          var rightbra = [')',']','}','\\right)','\\right]','\\right\\}'][pos];
        }
        result += res[0];
        result += rightbra;
        done = true;
      }
      if (node.type.startsWith('sqrt')) {
        result = '\\sqrt';
        result += res[0];
        done = true;
      }
      if (!done) {
        result = res[0];
      }
    }
    if (number_of_childs >= 2) {
      var binaryoperator = false;
      if (node.type.startsWith('plusminus') || node.type.startsWith('timesdivided')) {
```

```
            binaryoperator = true;
        }
        if (node.type.startsWith('power') || node.type.startsWith('sub')) {
            binaryoperator = true;
        }
        if (binaryoperator) {
            result = res[0];
            result += node.content;
            result += res[1];
            done = true;
        }
        if ((!done) && node.type.startsWith('frac')) {
            result = '\\frac';
            result += res[0];
            result += res[1];
            done = true;
        }
        if ((!done) && node.type.startsWith('nthroot')) {
            result = '\\sqrt';
            result += res[0];
            result += res[1];
            done = true;
        }
        if (node.type.startsWith('definite_integral')) {
            result = '\\int_';
            result += res[0];
            result += '^';
            result += res[1];
            result += res[2];
            done = true;
        }
    }
    if (done === false) {
        // handle bracket childs (maybe 1 or 2 or even more)
        var pos = -1;
        var count = 0;
        var temp = node.content;
        // Do not change node.content. Use temp instead.
        do {
            pos = temp.indexOf('§');
            if (pos > -1) {
                // console.log(node.id + ' ' + temp + ' ' + count + ' from ' + node.children);
                var left = temp.substring(0, pos);
                var right = temp.substring(pos + 1);
                var middle = res[count];
                // console.log(left + '::' + middle + '::' + right);
                temp = left;
                temp += middle;
                temp += right;
                count++;
            }
        } while (pos > -1)
        result = temp;
```

```
        }

        depth--;
        // console.log(node.id + '----------------------'.slice(0, 2 * depth) + result);
        // console.log('(' + depth + ') ' + result);
        return result;
    }
}
;

col = 0;
function paint_tree(tree, canvas, context) {
//     if (col === "#ffffdf") {
//         col = "#ffdfff";
//     } else {
//         col = "#ffffdf";
//     }
    col = "#ffffdf";
    context.fillStyle = col;
    context.fillRect(0, 0, canvas.width, canvas.height);
    context.font = '7pt Consolas';
    paint_tree_recurse(tree.root, tree.nodelist, -9999, -9999, 0, 0, context, 1);
}
;

function paint_tree_callback(currentNode, xa, ya, x, y, ctx) {
    console.log(currentNode.id + '::' + currentNode.children);
    console.log(xa + ' ' + ya + ' ' + x + ' ' + y);
    if (xa > -9999) {
//         var xf = 600;
        var xf = ctx.canvas.width / 2 - 100;
        var yf = 40;
        var xt = ctx.canvas.width / 2;
        var yt = 30;
        xxa = xa * xf + xt;
        yya = ya * yf + yt;
        xx = x * xf + xt;
        yy = y * yf + yt - 5;
        //console.log(xxa + ' ' + yya + ' ' + xx + ' ' + yy);
        ctx.beginPath();
        ctx.moveTo(xxa, yya);
        ctx.lineTo(xx, yy);
        ctx.stroke();
        ctx.fillStyle = "#5050ff";
        ctx.fillText(currentNode.type, xx + 2, yy);
        ctx.fillStyle = "#ff5050";
        ctx.fillText(currentNode.content, xx + 2, yy + 10);
    }
}
;

function paint_tree_recurse(currentNode, nodelist, xa, ya, x, y, ctx, factor) {
    paint_tree_callback(currentNode, xa, ya, x, y, ctx);
```

```
    var xa = x;
    var ya = y;
    factor = factor * 0.75;
    var cnchl = currentNode.children.length;
    for (var i = 0, length = cnchl; i < length; i++) {
        paint_tree_recurse(nodelist[currentNode.children[i]], nodelist, xa, ya, xa + factor * (i -
        0.5 * (cnchl - 1)), y + 1, ctx, factor);
    }
}
;
```