Roberto Noel
Gus Xia
Independent Study: MIR
9/24/19

Progress Report 1: Conceptualization

Introduction

On September 6, 2011, a counter complex user dubbed "Viznut" released a video showing how short, on-line C programs can generate waves which can be reproduced as music. Since then, inspired programmers have been experimenting with creating lines of code generating music. There have been many interesting findings which you can find here. One quite iconic symphony is the 42 melody, which was separately discovered by several people: "t*(42&t>>10)".

Objective

Since 2011, there has been some progress in understanding how these programs work, as well as some lines of code which take a more deterministic approach to generating music, take this "Rick Roll" for example. My goal with this project is leverage the power of machine learning to make new discoveries of programs with high-musicality. I will focus on the least understood versions of these programs, which are constrained to a certain operators and have less deterministic properties. The approach I will be taking is similar to the Ramanujan Machine, which uses meet-in-the-middle and Descent & Rappel to discover mathematical conjectures. Instead of using those algorithms however, I will design a genetic algorithm which will pick its "winning" children through Music Information Retrieval (MIR) techniques.

Approach

Before determining the MIR techniques that will be used to select children in the genetic algorithm, we first must have a systematic way of generating valid children (children that would actually compile in C. To do this, it is useful to use a data structure, which will allow for decomposition and re-composition of the parent. For this, I chose a binary-tree called an expression tree (see fig.1), in which the internal nodes are operators and the leaves are variables. The pre-order traversal of the tree should return the original expression.

This structure makes the creation of children easy, as we can go through each node and assign a probability of mutation depending on its type and depth in the tree. Consider the tree on fig. 1. As you can see, the nodes that are higher on the tree have a bigger over-all effect on the expression because they interact with more children on lower levels (especially true for multiplication, division and shifting). Thus probability of mutation should be lower at the top of the tree and higher at the bottom. Some operations should also mutate more easily than others, it may also be possible for certain expressions to "mate" and make children which share subtrees of the parents. These technicalities will be determined through experimentation as the algorithm is developed.

Roberto Noel
Gus Xia
Independent Study: MIR
9/24/19

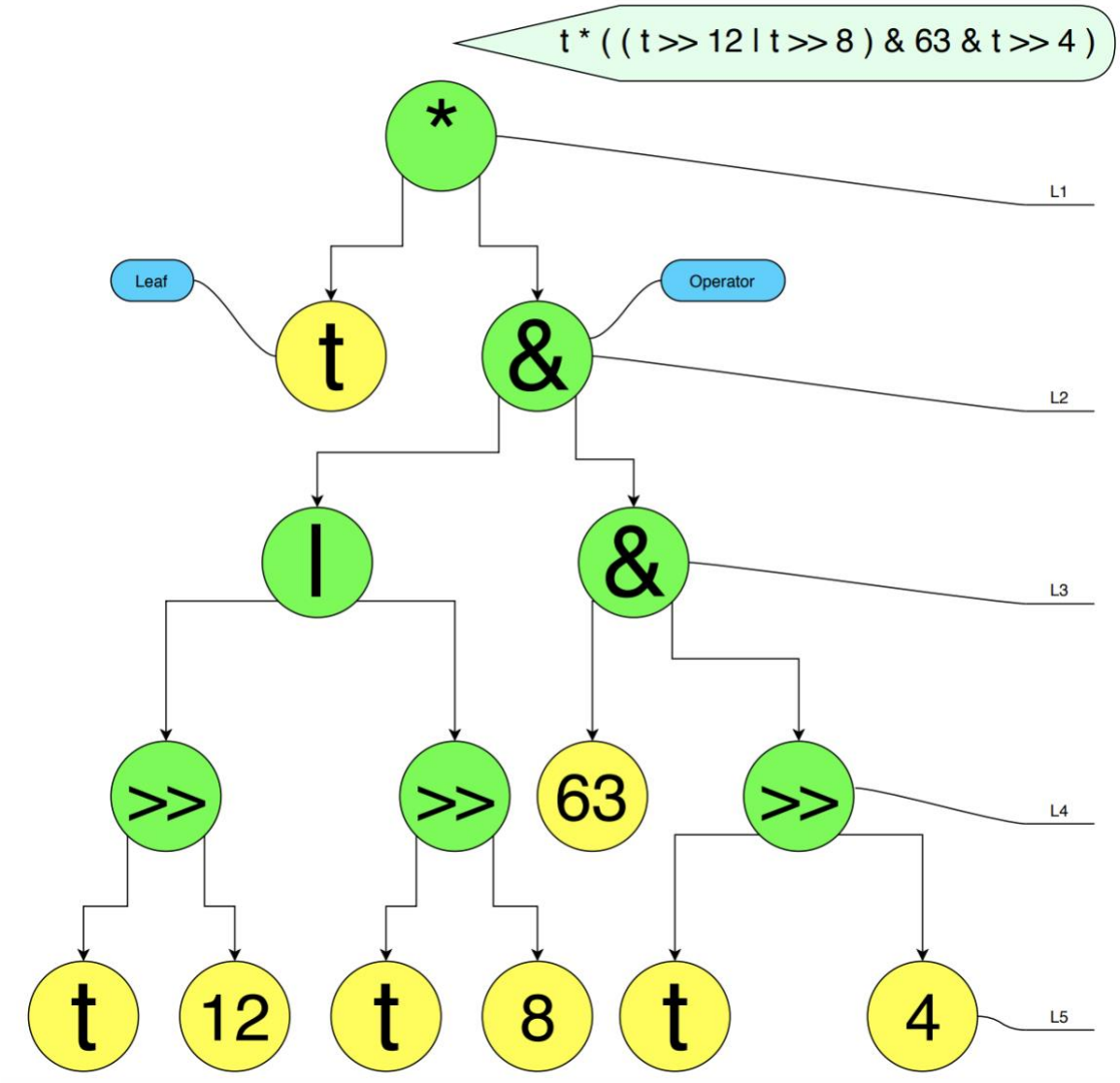

fig. 1