We propose to write a translator from L-System grammar specifications to Java classes. These classes must be fully compatible with an existing Java program which is designed for handling L-Systems (called Growth). This conversion will include automatic discovery of parameters to L-System symbols and parameters to the system as a whole, including appropriate name analysis of those symbols.

Growth already contains a complete framework of classes for the roles that particular symbols in L-Systems have. However, for each new grammar added to the program, six or seven small Java classes must be written by hand, and small changes made to an additional class to integrate the grammar with the existing code. Most of this task is extremely mechanical, so it seems like an excellent use for Eli.

The input to the translator is a complete L-System grammar and additional information about parameters to that grammar. Here is an example grammar:

A(length, width) → F(length, width) & (Angle to right) B(length * Length contraction to right, width * width contraction) / (180) & (Angle to left) B(length * Length contraction to left)
B(length, width) → F(length, width) + (Angle to right) $ B (length * Length contraction to right, width * width contraction) [- (Angle to left) $ B(length * Length contraction to left, width * width contraction)]
axiom → A(Initial length, Initial width)

Which builds “spiral” trees. Although this grammar seems a bit hard to read, this is the way that mathematicians working with L-Systems specify them. The translator will need to automatically recognize that A and B are distinct symbols, with A referencing B, that the axiom (first) symbol in the grammar is A, and a number of things about three different kinds of parameters. The first kind of parameter is a symbol parameter. “length” and “width” are the symbol parameters to both symbols in this grammar, but in other grammars different symbols may also have additional parameters. The scope of these parameters is the entire expansion of the symbol that they are parameters to, and no more (so, for example, they cannot be accessed in the “axiom” rule, and “length” refers to a different entity in A’s expansion than in B’s). The second kind of parameter is a L-System wide parameter. Every occurrence of these parameters is a defining occurrence, and the scope of each parameter is all of the rules. For example, every time “Angle to right” appears, it refers to the same entity. The final kind of parameter is a parameter to the whole Growth program, which is handled differently from other parameters. “width contraction” is a special constant of this type, and is not treated as a system-wide parameter.

Terminal L-System symbols are the following characters: ‘F’, ‘[’, ‘]’, ‘&’, ‘\’, ‘\’, ‘+’, ‘-’, ‘$’. As can be seen above, these terminal symbols may also take parameters (from 0 to 2), used to perform particular actions when actually drawing the L-System. For example, ‘+’ means “turn right” (in reference to the current position when drawing) by some number of degrees, given by the parameter to the symbol. In Growth, every symbol in the grammar must be represented by a class. Terminal symbols with different parameters (for exam-
ple, the two instances of ‘&’ above) are represented by different classes (so to ‘&’ classes would be created). Terminal symbols have no expansion.

L-System wide parameters in Growth are handled as static object instances (for design reasons not important here). This means that some Parameter objects may need to be available to multiple types of symbols, like “Angle to right” above, which is needed by both an instance of ‘+’ and an instance of ‘&’. Growth handles this via inheritance. In this case both ‘+’ and ‘&’ (one instance of ‘&’, since it takes two different parameters) would extend a single symbol class containing the static parameter. This class would be abstract. Parameters must be created with certain attributes: a name, the same name as used above, a type (types are either continuous or discrete; continuous parameters additionally must specify whether the value of the parameter will be meaningful to the user), a range that they may hold, inclusive, and an initial value. Since all but the name are not specified in the grammar itself, the input language will also need to specify these attributes for parameters named in the grammar.

A set of Java classes which compile and specify the same L-System as the grammar above, within the Growth system, is considered correct output. Clearly names of classes and variables will not be as legible, but the original grammar can certainly be considered part of the documentation to negate this difficulty. The translator is expected to produce structurally identical (or extremely similar) code. Note that all systems have a “Number of Iterations” parameter in their axiom symbol; this does not need to be specified anywhere in the input language since it is known. Below is a (partial) human translation of the example grammar.

```java
package generator.symbols.aono;

import generator.ContinuousParameter;
import generator.DiscreteParameter;
import generator.Parameter;
import generator.ValueHiddenContinuousParameter;
import generator.symbols.AxiomSymbol;
import generator.symbols.Symbol;

public class AonoAxiomSymbol extends AxiomSymbol {
  private static Parameter[] params;
  private static final int ITERATIONS = 0;
  private static final int LENGTH = 1;
  private static final int WIDTH = 2;
  static {
    DiscreteParameter iters = new DiscreteParameter(1, 10, 6,
      "Number of Iterations");
    ContinuousParameter len = new ValueHiddenContinuousParameter(0.2, 1,
      0.7, "Initial Branch Length", 2);
    ContinuousParameter wid = new ValueHiddenContinuousParameter(0.01, 1,
      0.15, "Initial Branch Width", 2);
    params = new Parameter[] {iters, len, wid};
```
public Symbol[] getExpansion()
{
    double l = ((ContinuousParameter) params[LENGTH]).getDoubleValue();
    double w = ((ContinuousParameter) params[WIDTH]).getDoubleValue();
    return new Symbol[] {new AonoTrunkSymbol(l, w)};
}

public Parameter[] getParameters()
{
    return params;
}

protected int getIterations()
{
    return ((DiscreteParameter) params[ITERATIONS]).getIntValue();
}

public Symbol[] getParameterSymbols()
{
    return new Symbol[] {new AonoAxiomSymbol(), new AonoTrunkSymbol(0, 0),
    new AonoTurnLeftSymbol()};
}

---

generator.symbols.aono;

generator.symbols.BaseSymbol;
generator.symbols.PopSymbol;
generator.symbols.PushSymbol;
generator.symbols.Symbol;

class AonoTrunkSymbol extends AonoMoveForwardSymbol
{
    public AonoTrunkSymbol(double len, double wid)
    {
        super(len, wid);
    }

    public Symbol[] getExpansion()
    {
        Symbol f = new BaseSymbol(length, width);
        Symbol b1 = new AonoBranchSymbol(length *
(1 - params[LENGTH_CONTRACT_LEFT].getDoubleValue()), width *
super.getWidthContraction());
        Symbol b2 = new AonoBranchSymbol(length *
(1 - params[LENGTH_CONTRACT_RIGHT].getDoubleValue()), width *
super.getWidthContraction());

        return new Symbol[] {f, new PushSymbol(), new AonoPitchLeftSymbol(),
    b1, new PopSymbol(), new AonoRollSymbol(), new PushSymbol(),
    new AonoPitchRightSymbol(), b2, new PopSymbol()};
    }
}
public abstract class AonoRightChangeDirectionSymbol extends ChangeDirectionSymbol {
    protected static ContinuousParameter[] params;

    protected static final int ANGLE_RIGHT = 1;

    static {
        ValueShownContinuousParameter l1 =
            new ValueShownContinuousParameter(0, 90, 10, "Angle to Left", 1);
        l1.setTicks(15, 5);
        params = new ContinuousParameter[] {l1};
    }

    public Parameter[] getParameters() {
        return params;
    }
}

package generator.symbols.aono;
import generator.Action;
import generator.PitchAction;

public class AonoPitchRightSymbol extends AonoChangeDirectionSymbol {
    public Action getAction() {
        return new PitchAction(params[ANGE_RIGHT].getDoubleValue());
    }
}

package generator.symbols.aono;
import generator.Action;
import generator.TurnAction;

public class AonoTurnRightSymbol extends AonoChangeDirectionSymbol {
    public Action getAction() {
        return new TurnAction(params[ANGE_RIGHT].getDoubleValue());
    }
}
return new TurnAction(-params[ANGLE_RIGHT].getDoubleValue());