2011 年“种子杯”
编程 PK 赛
复赛说明文档

SLogo

普通青年

2011/11/11

目录

1 程序编译运行环境

处理器	Intel® Core™2 Duo P8700
主频	2533MHz
内存	4.00 GB
操作系统	Microsoft® Windows 7 SP1 64-bit
使用语言	C#
编译器	Microsoft® Visual C# 2008
运行环境	.NET Framework 3.5

2 第三方依赖库

名称	项目主页
Fast lightweight expression evaluator	http://flee.codeplex.com/
Svg rendering engine	http://svg.codeplex.com/
SVGLib	http://www.codeproject.com/KB/cs/svgpad.aspx
Command Line Parser Library	http://commandline.codeplex.com/

3 程序结构

程序结构及运行的大致流程如下：

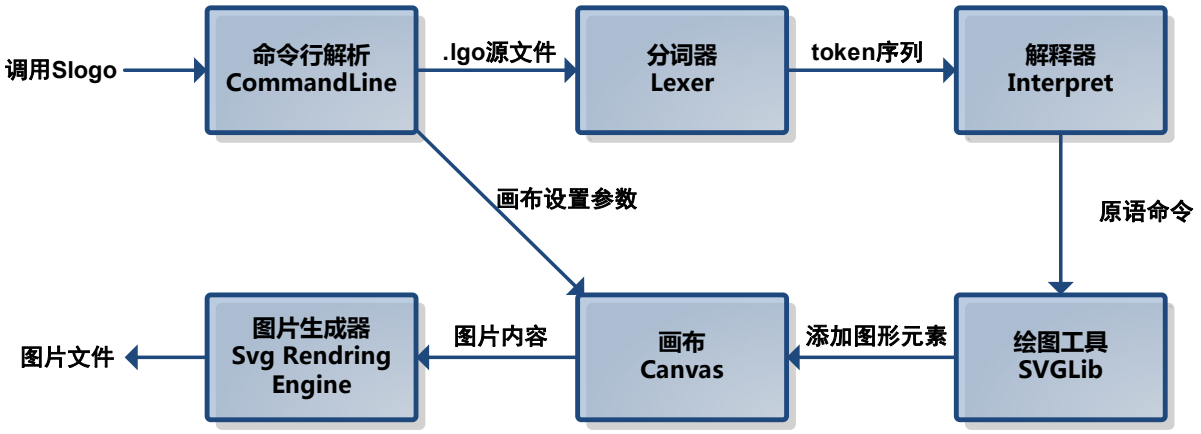

Figure 1 程序结构图

3.1 命令行解析

用来处理调用 SLogo.exe 时的参数， 验证参数正确性，并将[filename]参数送入到解释器模块中，将[-m margin]、[-T type]等画布设置参数送入到画布模块中。

参数解析使用了 GNU Getopt（http://www.gnu.org/s/hello/manual/libc/Getopt.html）的语法，因此各选项的顺序可以任意调换，下列输入也是等效的：

```
SLogo –m 10 path/to/file
SLogo –m10 path/to/file
SLogo –m=10 path/to/file
SLogo --margin 10 path/to/file
SLogo --margin=10 path/to/file
```

3.2 分词器

分词器接受一个文件流输入，读取流中的文本数据进行分词（Tokenize）。本程序的分词器混合使用了条件转移和正则表达式来完成分词操作。

分词器生成下列种类的词语（Token），每个词语中还附带了其行号与列号信息，方便给出调试信息：

1. 未知字符：@#$?等单个未知字符；
2. 标识符：_im_identifier0 这样的下划线、字母和数字组成的标识符；
3. 整数；
4. 浮点数；
5. 字符串：双引号包围起来的包含空格的字符串，可以使用\\和\”进行转义；
6. 空白符：空格与制表符；
7. EOF：文件尾；
8. EOL：换行符，支持 DOS/Windows，Linux 与 Mac 三种不同形式的换行符；
9. 变量名：以冒号开始的合法标识符；
10. 运算符：圆括号、加减乘除等运算符；
11. 方括号：用于 if、ifelse、repeat 的方括号，方括号间是合法的代码块。

3.3 解释器

读取脚本，对 LOGO 语句进行解释，判断源代码中的语法错误，将符合语法规定的源代码转换成原语命令，调用画布模块执行基本绘图命令以及附加功能中的画笔参数设置。

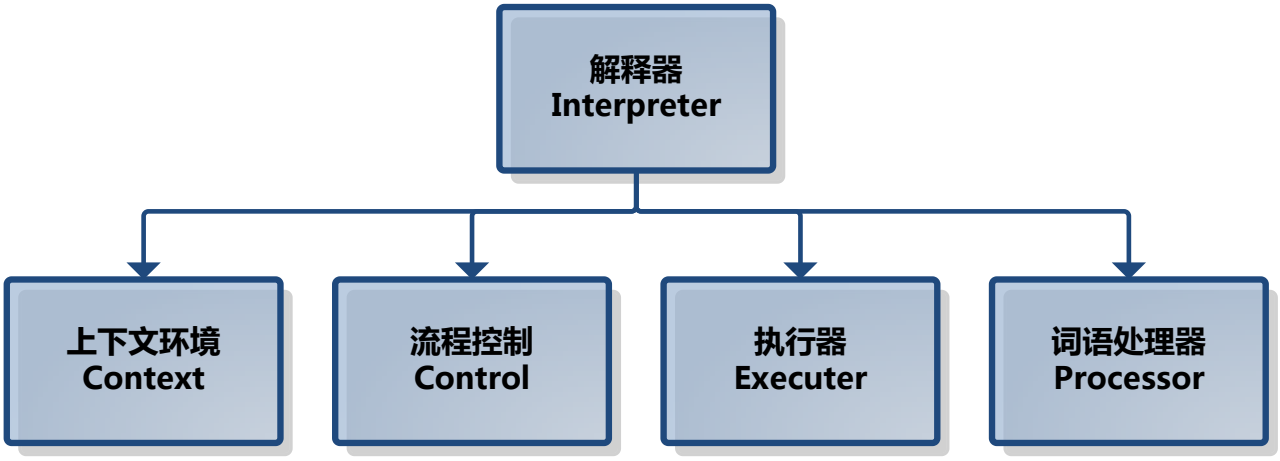

Figure 2 解释器的结构

3.3.1 上下文环境

我们将程序当前运行位置的变量集合称为上下文环境（Context）。解释器需要追踪、提供与维护这一变量集合。尤其是在自定义子流程中，同名的局部变量优先于全局变量。在 SLogo 解释器的实现中，使用了集合作为数据结构提供上下文环境。

3.3.2 流程控制

Logo 语言中有简单的流程控制功能，对应的语法分别是 if、ifelse 和 repeat。为了实现流程控制，解释器需要记录执行跳转的位置等数据。

3.3.3 执行器

执行器负责两种语句的执行：原语命令与自定义子流程。对于原语命令，比如 FD 和 RT，执行器的工作是检查参数并执行相应的 Native Code 方法。对于自定义子流程，执行器将检查参数、添加实参作为局部上下文环境，跳转进入子流程代码段，捕获 STOP 发出的信号以及清理现场。

3.3.4 词语处理器

处理器（Processor）的作用是对分词器生成的一串有序的词语序列进行语法分析。它维护一个游标，在词语链表（Token Linked List）中移动并以定义好的语法进行解释。

例如，对于简单的 ifelse 语句，解释过程如下：

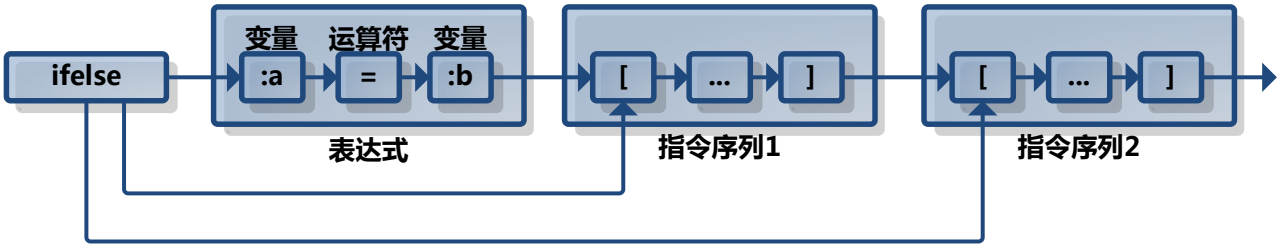

Figure 3 ifelse 的分析

如图所示，处理器遇到关键字 ifelse 后，先向后扫描并匹配得到表达式、指令序列 1 和指令序列 2。这三个部分作为部分链表，可以用同样的方式送入处理器中进行二次处理。首先解释表达式，若为真，将指令序列 1 处理器跳转处理该部分链表；若为假，同理处理指令序列 2 所在的部分链表。

从另一个角度来说，一个 ifelse 语句可以看成类似 Lisp 语言中这样的嵌套表：

```
(ifelse (= 'a 'b) (……) (……))
```

在 SLogo 的解释器实现中，一个中括号包围的指令序列实际上被看成一整个源代码正文，可以以相同的方式被解释。

3.4 画布

保存并维护画布及画笔的设置参数，以及海龟的属性，执行基本绘图命令，并将绘图操作转换为矢量图的各元素参数，添加到图形生成器中。

3.5 图形生成器

根据矢量图的各元素参数生成 svg 图片的 XML 代码，并根据需要将.svg 格式源码转码为.png 格式。当设置了 margin 参数时，维护当前图片的实际占用大小，并在最后对图片进行裁剪。

4 算法原理

4.1 源代码解释

首先通过语法分析器把源代码中的单词（Token）找出来，语义分析器再把这些分散的单词按预先定义好的语法组装成有意义的表达式、语法、过程等等，同时进行语义检查，进行简单的错误处理。

然后解释器每次读入一条语句，通过表达式运算器处理数值表达式，将其转换为数值(或布尔值)，并把循环、自定义过程等复杂指令解释为简单的原语指令的组合，同时进一步检查代码的正确性，最后将原语指令送到执行器中执行。

4.2 错误处理

由于 Logo 语言为解释型语言，每个语句都是执行的时候才翻译，所以一般情况下是执行到错误的语句后就提示错误并退出。为了能让解释器发现更多的错误，遇到一个错误后，程序会对后面的语句继续进行语法检查。

4.3 图形裁剪

如果用户输入了可选参数-m，则在绘制图形时，每添加一个元素，便会根据添加元素所占范围的坐标值维护当前图片实际占用的范围，最后生成图片前再针对图片实际占用的范围及 margin 值对所有元素进行平移，并修改图像大小。

4.4 文本框范围确定

为了能确定文本框实际占用的范围，我们采用了等宽的字体进行输出，并根据字体大小及字数，确定文本框的长宽，然后由海龟当前位置及方向计算出文本框的四个角的坐标，对当前图片的实际占用范围进行维护，以下便是测试效果（margin = 0，为了方便观察，添加了背景色）

Figure 4 文本框裁剪效果

4.5 图形生成

在图形生成的过程中，我们使用了两个第三方库，分别是修改过的 SvgLib 和 Svg Rendering Engine。

其中，SvgLib 用来生成 svg 文件， 我们只需要对文档对象模型（DOM）进行操作， SvgLib 便会生成对应的 XML。由于该库的版本老旧，不支持诸如变换之类的功能，我们对其源码进行了一定程度的 Hack 已适应自己的需要。而 Svg Rendering Engine 是是一个 Svg 渲染引擎。更确切地说，是一个栅格化引擎（Rasterize engine），通过该渲染引擎，我们可以将 svg 转换为位图格式(Bitmap)，然后将位图转换成 png 编码的图像。

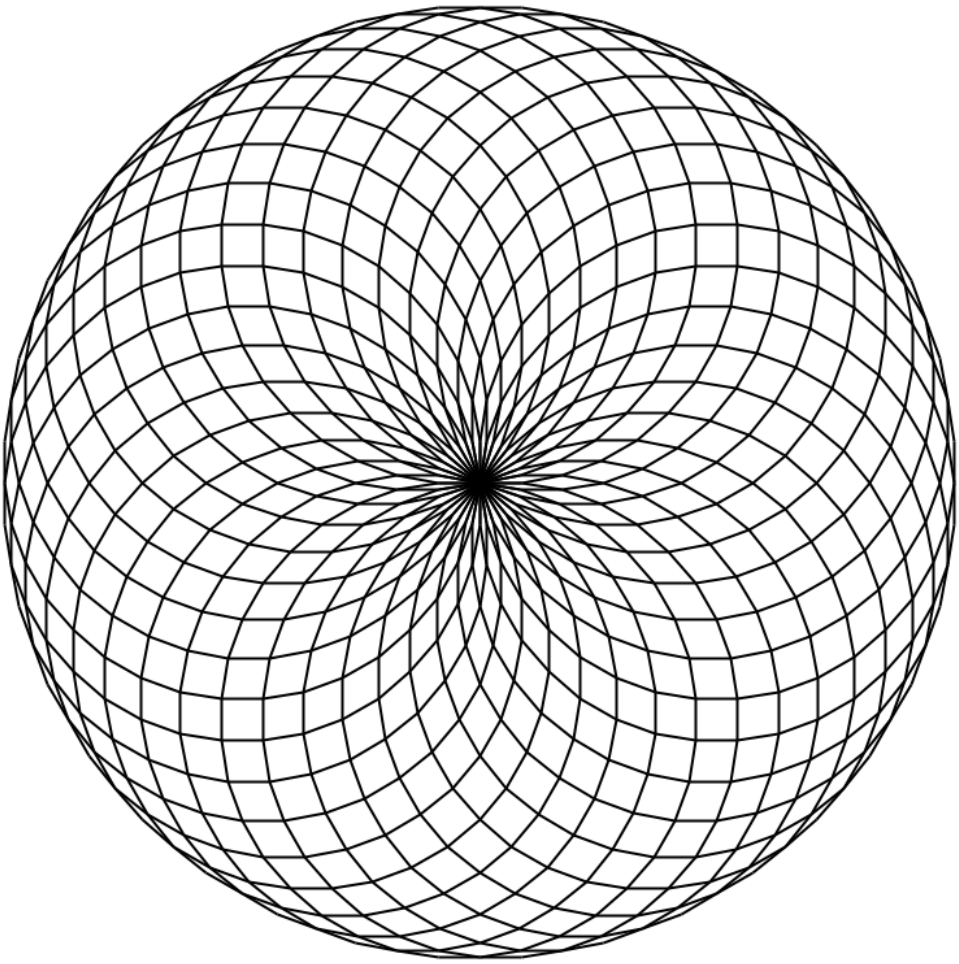

Figure 5 通过 Svg Rendering Engine 渲染的一个 Logo 图形

5 特别设计

1. 一条命令直接画圆、椭圆、矩形（圆角矩形）、贝塞尔曲线等，具体命令如下：

命令格式	命令功能
Circle :r	以海龟位置为圆心，r 为半径画圆
Ellipse :rx :ry	以海龟位置为中心，rx、ry 分别为 x、y 方向上的半轴长画椭圆
Rect :w :h :r	以海龟位置为左上角顶点，w、h、r 分别为长、宽、圆角半径画矩形。 若 r 为 0，则画出来的图形为普通的矩形
Bezier :x1 :y1 :x2 :y2	以海龟当前位置为起点，(x2，y2)点为终点，过(x1，y1)点做二次贝塞尔曲线

2. 画布参数设置（颜色、透明度等），具体命令如下：

命令格式	命令功能
Color :r :g :b	设置边框颜色，r、g、b 分别为 RGB 分量（0~255）下同
Fill :r :g :b	设置填充色
Background :r :g :b	设置背景颜色
Stroke :w	设置线条宽度为 w，范围是 1~20
Opacity :o	设置线条不透明度为 o（0~1）
FillOpacity :o	设置填充色不透明度为 o（0~1）

3. [-m margin]可接受 1~4 个参数，四方向边距按 CSS 规则定义：

例子：

-m 10 → 每边边距都为 10px

-m 10,20 → 上下边 10px 边距，左右边为 20px 边距

-m 10,20,30 → 上边 10px，左右边为 20px，下边为 30px

-m 10,20,30,40 → 按照上、右、下、左的顺序应用边距 10px，20px，30px，40px

注意！指定边距的逗号间不能有空格！

4. 准确的错误提示，包括行号、列号及错误信息，对于语法错误，会检查所有的错误，而不是遇到第一个错误就退出。

5. 支持更复杂的逻辑运算符：包括小于等于（<=）、大于等于（>=）、不等于（<>），以及布尔逻辑运算符：与（and）、或（or）、非（not）、异或（xor）。

6. 支持更复杂的数学运算，包括乘方运算（^）、取模运算（%），以及数学函数 sin(x)，cos(x)，cot(x)，tan(x)，pow(x, y)，abs(x)，round(x)，sqrt(x)，floor(x)，ceiling(x)，exp(x)，log10(x)，log(x)。

7. 部分附加功能效果展示：

Figure 6 附加功能效果

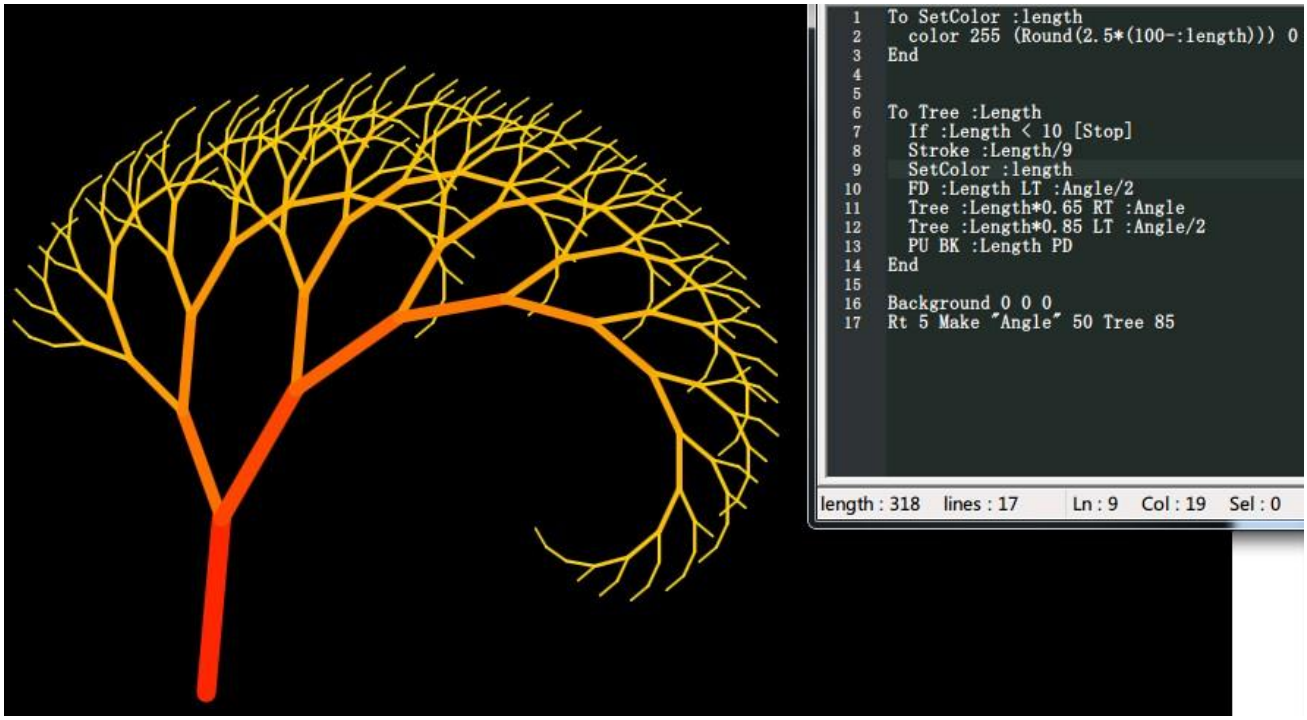

Figure 7 附加功能效果

6 心得体会

我们原本认为本次复赛的题目会像往届的复赛题一样，主要考察程序的执行效率，但这次的题目有点出乎我们意料，可以说是初赛题目的升级版吧，命令解析的任务更加复杂了，而且还需要绘制图形。

其实整个程序的关键就是对 Logo 语言的解释，这也是程序最复杂的部分，由于还没有开始学编译原理，所以开始阶段实际上是非常困难的，我们几乎下载所有了开源 Logo 软件进行研究，但是满足功能需求的 Logo 软件源代码都非常复杂，整整两天我们都在尝试移植其中的某一款软件，均未成功。

最终我们还是决定自己进行设计，我们通过查阅编译原理的相关资料，开始一点一点的尝试，从词法分析、语法分析、表达式求值，到目标代码的生成，直到作品提交截止期前的两天，我们才完成了 Logo 语言的解释模块以及执行模块，开始与已经编写好的绘图模块进行联调。

现在回想起来，虽然最开始看到题目感觉非常困难，但通过自己的努力和团队的合作，亲身尝试了以后，发现整个开发过程其实也没有想象中的那么艰难。而且这样的一次挑战对我们团队来说也是一次很好的锻炼。