# Automating Code Magnet Generation

Julia Dana

## Contents

## Todo list

# 1 Introduction

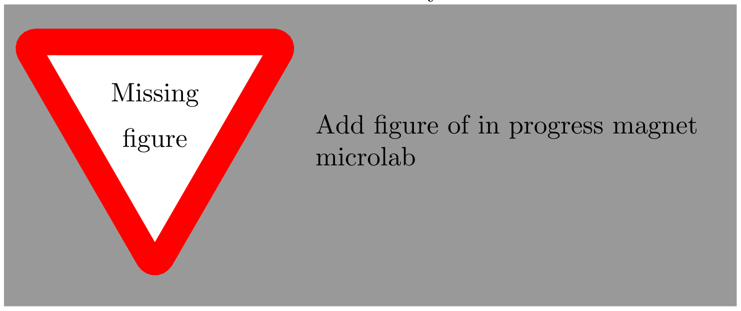
## 1.1 The Overview: Easier Creation of Code Magnet Microlabs

The purpose of this project is to assist in the creation of code magnet microlab assignments for WAGS by creating magnets from a completed solution file. This is accomplished in a manner that supports multiple programming languages, and allows additional languages to be added with minimal configuration. Additional tools that support this idea of easier creation of assignments are also included, such as an automated interaction with the WAGS website to create assignments. Also, this project defines new formats for the specification of magnets, including object, JSON and YAML.

Improve this sentence

## 1.2 The Context: What is WAGS?

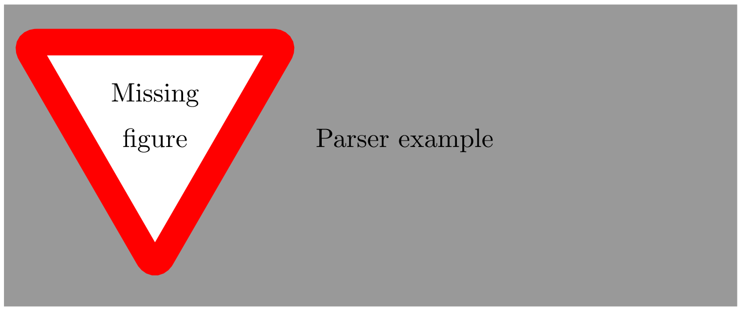
WAGS (Web Automated Grading System) is an ongoing project of Appalachian State University. It is an online tool for microlabs. Microlabs are short, 5-10 minutes hands-on activities that are intended to be done as a part of a regular (i.e. not lab) class session to reinforce the concepts that are currently being covered. There are multiple types of microlabs provided by WAGS, but the one that this project is interested in is code magnet microlabs. These are microlabs where the student is given pieces of code (code magnets) and has to choose the correct ones and order them correctly to "write" a solution to the microlab.



Add figure of in progress magnet microlab

## 1.3 The Problem: Brittle Input

The problem is that creating these magnet microlab assignments on the WAGS website is a somewhat painful process. The current parser creates a magnet per line, and this can result in having to format the input file so that is no longer the same as a solution file, and is indeed no longer valid for its language.



If your input cannot be handled by this brittle parser. WAGS does provide a manual input for magnets. However, using this manual input requires magnets to either be entered one at a time to the magnet creation wizard, or for the user to directly type the final magnet (including HTML escape sequences) to the parsed output section.

### 1.4 The Solution: Parsing by Grammar

The solution to this is to use a more robust parser that is based on the grammar of a language, rather than simply splitting on line breaks. This project does this by using the ANTLR4 parser generator[2] and freely available grammars for common languages[1].

tokens - words grammar rules - phrases - match an ID with one or more productions (pp 65 and 66)

Discuss grammars

One of the types of output from a parser is a parse tree

Discuss parsing and syntax trees

A parser generator takes the description of the language given in a grammar, and creates the parser for that language. There are many such parser generators, including . This project uses ANTLR4, which generates parsers in Java. It is an LL(*) parser generator.

Describe parser generators (using ANTLR4 as example)

list and cite parser generators

expand on LL(star)

## 2 Development results and future extensions

### 2.1 What It Does: Internal Functions

#### 2.1.1 Parses by Grammar

The magnetizer takes an input file, and parses it with the parser generated by

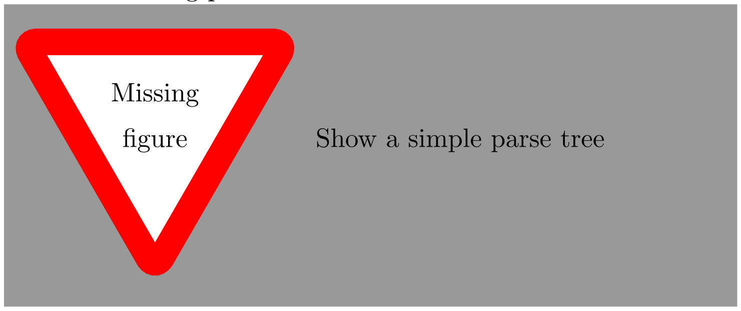
Clarify this entire section

magnetizer? find a better

ANTLR from the grammar specification. The result of this parsing is a parse tree, which can be represented graphically. An example follows.

For this simple Java file, Hello.java

The resulting parse tree is:



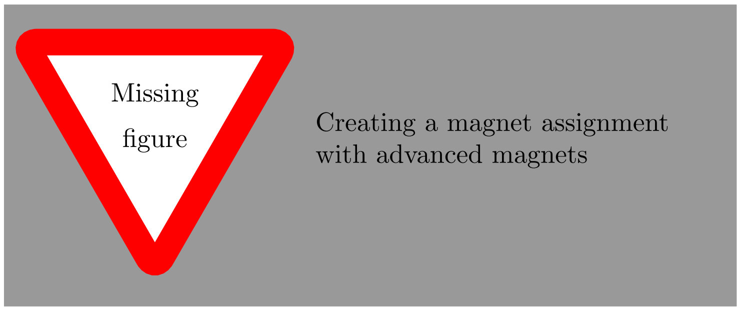
Each language supported by the magnetizer has an entry in a configuration file that specifies which nodes in the parse tree should trigger the creation of a magnet. Whenever a child node triggers its own magnet, the parent is given a drop zone for that section, and that section of text is not directly included in the parent magnet.

ANTLR visitor vs. listener

### 2.1.2 Alternative Magnets and Other Instructor Directives

### 2.1.3 Improved Represention of Magnets

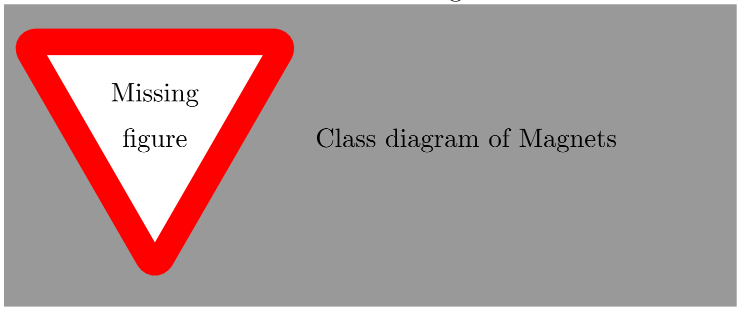
The old format for describing magnets is difficult to read and worse to try to manually type. Some of its quirks are that magnets are separated by an unusual separator (`.:|:.`), areas that accept other magnets (aka drop zones) are indicated by a seemingly random set of HTML tags (`<br><!-- panel --><br>`), and any special characters used in the code magnet must be escaped for HTML (so something like `1 < 2` would need to be entered as `1 &lt; 2`). This format also is limited when trying to create more advanced types of magnets. The WAGS system addresses this problem by adding more form fields when adding "advanced Java magnets", but this does not work well for adding additional languages.



Creating a magnet assignment with advanced magnets

This project creates an underlying data structure for magnets, as shown in figure . This structure can then be serialized as desired. Currently, serialization to the old magnet format is supported for backward compatibility, as well as JSON and YAML for more robust usages.

link figure of class diagram



Class diagram of Magnets

## 2.2 What It Does: External Functions

### 2.2.1 Command Line Tool

### 2.2.2 Automated Interaction With WAGS Website

## 2.3 What It Could Do: Future Extensions

### 2.3.1 Automatic Generation of Alternative/Distractor Magnets

The next step in easily creating code magnet assignments is to have the magnet creation tool not simply create magnets needed for the correct solution and any additional magnets specified by the instructor, but also to automatically create appropriate alternative "distractor" magnets for common student misconceptions and errors. This level of manipulation is available because we have the whole parse tree to work with during magnet creation, but would require significant work per language to define.

### 2.3.2 GUI Tool

A GUI tool that can open an input file, perform the actions of the CLI tool on it, and also has an editor to assist in the addition of any instructor directives desired would be a nice addition to this project.

#### 2.3.3 Supporting New Languages

This project currently can create magnets for Python3 and Java, however it is set up to easily allow the addition of new languages. Basically, this is done by finding and adding an ANTLR4 grammar file for the desired language, making sure it conforms to a handful of guidelines, writing a short configuration file, and running a setup action on the project. Full details of this process are in section 4.5 The Expansion: Adding a New Language.

#### 2.3.4 Additional Serialization Formats

Because magnets are now objects, additional serialization formats (such as XML) could easily be added.

# 3 User Guide

## 3.1 Quick Start - CLI Tool

The primary interface to this project is the magnetizer command line tool. This tool is capable of outputing to WAGS-style magnets that can be copy-pasted into the parsed sections of the WAGS website or to a JSON or YAML representation of the magnets.

Make sure usage is the latest version.

```
Usage: magnetizer [options] file
    -l, --language LANGUAGE          Specify a language (default: Java)
        --json                       Print the JSON output
        --yaml                       Print the YAML output
        --[no-]wags                  Print the output as WAGS magnets
(default)
    -o, --output-file BASE_FILENAME  Output to a file
    -h, --help                       Show this message
```

## 3.2 Installation for Use

## 3.3 Explanation of Output

#### 3.3.1 Java

The current configuration for Java creates magnets for

explain this in a way that doesn't require deep understanding of the ANTLR grammar.

- Package declarations
- Import declarations
- Type declarations
- Class body declarations - this is the nonterminal that includes anything that can be directly in the class body

- Block statements - this is the nonterminal that includes anything that can be directly inside a block.

### 3.3.2 Python 3

The current configuration for Python creates magnets for

explain this in a way that doesn't require deep understanding of the ANTLR grammar.

- simple statements
- compound statements

## 3.4 Automatic Upload to WAGS Website

## 3.5 Modifying the Output: Instructor Directives

Instructor directives are information that can be put in the input file that change how the magnets are created. They are a special version of comments. The triggering syntax varies per language, but the directives are the same for all languages.

**Java** `/*# <directive> */`

**Python3** `## <directive>`

Note that directives in python are somewhat limited because comments cannot occur in a line.

Verify that there are no inline comments in python

### 3.5.1 Creating Alternative Magnets: ALT and ENDALT

The ALT and ENDALT directives are used to create alternatives with different text for a magnet. The ALT directive also takes the desired alternative text, and the ENDALT directive is used to indicate where the end of the alternative text is. These directives should always be used as a pair, and encompass the text that should be replaced. They cannot go around anything that would trigger the creation of a new magnet or drop zone.draft

Concrete example of ALT

### 3.5.2 Supressing Drop Zones: NODROP

The NODROP directive suppresses the creation of drop zones for the following magnet. An example of when this would be used is to put an entire loop on a single magnet, rather than having the individual statements in the body be on their own magnets.

Concrete example of NODROP

# 4 Developer Notes

## 4.1 The Environment: Languages and Libraries Used

Section needs more details

This project is written to run on JRuby. The magnetizer itself is written in Ruby, but it uses the Java classes provided by ANTLR. JRuby allows this to occur.

Automated interaction with the WAGS site is provided by using Capybara to interact with Selenium (drives Firefox) or Poltergeist/PhantomJS (headless).

Rake (Ruby make) is used to process new grammar files and other build tool functionality.

RSpec is used to handle testing

### 4.1.1 Installation for Development

Consider if this section should be on its own

## 4.2 The Testing

## 4.3 The Design

## 4.4 The Configuration: Specifying Magnet Sections

## 4.5 The Expansion: Adding a New Language

### 4.5.1 New Grammar Specification

A new G4 file can be added to the system. However, whitespace is require to go on channel 1, or your magnets will not have any whitespace, and you might get things like “publicclassMyClass”.

Final export of bibliography from Mendeley and link it

Make sure citations and bibliography are styled correctly

## References

[1] antlr/grammars-v4.

[2] Terence Parr. *The Definitive ANTLR 4 Reference*. The Pragmatic Programmers, 2012.