Custom Programming Language: Wizardry

Ronnie Yalung | CS 3252

Steps to Success

1) Create a grammar/syntax for the custom language

2) Use syntax to create a recursive descent parser (RDP)

3) Create a transpiler for code generation using the parser

Purpose

How does this link to our class?

- A programming language is practically a formal language defined by syntax (grammar)

- The parser is essentially a PDA

- The code generator/transpiler reflects:
  - The **decidability** of ensuring syntax and semantics match your grammar rules
  - The **computability** of transforming one representation of computation into another

Background / Research

In class, we have not made any RDPs (through programming/coding)

Unfortunately, ChatGPT is horrendous at creating any type of RDP.

- Learned from Dmitry Soshnikov 18 video course: Youtube , Online-course

ChatGPT is however not too shabby with helping create a transpiler

Step 1: Grammar/Syntax

Deliverables:

- Unique, new
- Allows:
  - Variable declaration & assignment
  - Basic data types: String, Number, Boolean
  - Arithmetic operations
  - Conditional statements
  - Loops (while loop)
  - Functions & function calls

Step 1.01: Naming the language

Wizardry

- ‘Magical’ syntax, coding with it feels like casting a spell

Step 1: Grammar/Syntax

Example Code:

```
conjure mana bind 10 <~>

enchant replenish(resource) {
    lest (resource mirrors 0) {
        resource bind resource imbue 10 <~>
    } fallback {
        resource bind resource deplete 10 <~>
    }
}

conjure empty bind 0 <~>

conjure spellcast bind mana forsakes
empty <~>
```

```
'bind': '=', // assignment
'imbue': '+', // addition
'deplete': '-', // subtraction and unary negative
'amplify': '*', // multiplication
'split': '/', // division
'prevails': '>', // greater than
'falters': '<', // less than
'mirrors': '===', // equal to
'forsakes': '!==', // not equal to
'whilst': 'while', // while loop
'enchant': 'function', // function declaration
'bestow': 'return' // return
<~>: ';' // delimiter
```

Step 2: Recursive Descent Parser

Deliverables:

- Writes parser functions corresponding to the grammar rules
- Building an abstract syntax tree (AST) to represent the structure of the parsed code
  - *AST needed for translation into executable code

*using Javascript

Step 3: Code Generator / Transpiler

Deliverables:

- Translates the AST into valid executable Javascript code
- Writes said code into a file for it to be executed in

EXAMPLE RUN (1):

tests > example2.lt

```
conjure mana bind 10 <~>

enchant replenish(resource) {
    lest (resource mirrors 0) {
        resource bind resource imbue 10 <~>
    } fallback {
        resource bind resource deplete 10 <~>
    }
}

conjure empty bind 0 <~>

conjure spellcast bind mana forsakes empty <~>
```

Code from /tests/example2.lt

bin > letter-rdp.js > main

```
#!/usr/bin/env node

'use strict';

const {Parser} = require('../src/Parser');

const fs = require('fs');


function main(argv) {
    const [_node, _path, mode, exp] = argv;

    const parser = new Parser();

    let ast = null;

    // Direct expression:
    if (mode === '-e') {
        ast = parser.parse(exp);
    }

    if (mode === '-f') {
        const src = fs.readFileSync(exp, 'utf-8');
        ast = parser.parse(src);
    }

    console.log(JSON.stringify(ast, null, 2));
}

main(process.argv);
```

/bin/letter-rdp.js

/src/Parser.js

```
ronnie.yalung@Ronnies-MacBook-Pro PROJECT % ./bin/letter-rdp.js -f tests/example2.lt
{
  "type": "Program",
  "body": [
    {
      "type": "VariableStatement",
      "declarations": [
        {
          "type": "VariableDeclaration",
          "id": {
            "type": "Identifier",
            "name": "mana"
          },
          "init": {
            "type": "NumericLiteral",
            "value": 10
          }
        }
      ]
    },
    {
      "type": "FunctionDeclaration",
      "name": {
        "type": "Identifier",
        "name": "replenish"
      },
      "params": [
        {
          "type": "Identifier",
          "name": "resource"
        }
      ],
      "body": {
        "type": "BlockStatement",
        "body": [
          {
            "type": "IfStatement",
            "test": {
              "type": "BinaryExpression",
              "operator": "mirrors",
              "left": {
                "type": "Identifier",
                "name": "resource"
              },
              "right": {
                "type": "NumericLiteral",
                "value": 0
              }
            },
            "consequent": {
              "type": "BlockStatement",
              "body": [
                {
                  "type": "ExpressionStatement",
                  "expression": {
                    "type": "AssignmentExpression",
                    "operator": "bind",
                    "left": {
                      "type": "Identifier",
                      "name": "resource"
                    },
                    "right": {
                      "type": "BinaryExpression",
                      "operator": "imbue",
                      "left": {
                        "type": "Identifier",
                        "name": "resource"
                      },
                      "right": {
                        "type": "NumericLiteral",
                        "value": 10
                      }
```

**full ast too large to display*

EXAMPLE RUN (2):

Copy and paste the full AST into the ast variable at the top of the /transpiler/transpile.js file:

run the transpile.js file

Congrats, you're done! Javascript code that matches the Wizardry code you typed is stored in: */run/transpiledCode.js*

*Javascript code produced should contain no errors, be syntactically valid, and have the ability to run.

EXAMPLE RUN (3): Final Comparison

Wizardry Code:

```
conjure mana bind 10 <~>

enchant replenish(resource) {
    lest (resource mirrors 0) {
        resource bind resource imbue 10 <~>
    } fallback {
        resource bind resource deplete 10 <~>
    }
}

conjure empty bind 0 <~>

conjure spellcast bind mana forsakes empty <~>
```

Resulting JS code:

```
let mana = 10;

function replenish(resource) {
if (resource === 0) {
resource = resource + 10;
} else {
resource = resource - 10;
}
}

let empty = 0;

let spellcast = mana !== empty;
```

DEMO

To get to the correct state in your terminal, use the following commands: (Assuming you are in the directory of your project):

note: make sure you have node.js installed

- `chmod +x bin/letter-rdp.js` // Makes the file a valid executable
- `npm init -y` // Initialize a package. json file in directory

Now we can run commands from the letter-rdp file such as:

- ./bin/letter-rdp.js -e '2 imbue 2 <~>' // lines of code
- ./bin/letter-rdp.js -f tests/example.lt // file with Wizardry code

DEMO contd.

Running said commands from the previous slide yields an abstract syntax tree in the terminal

1) Copy and paste the AST into the ast variable at the top of the transpiler/transpile.js file

2) Run the transpile.js file

3) Done! Valid executable code can be found in the /run folder