CS F363 Compiler Construction, Sem II 2020-21, Complete Assignment Specs

Ramprasad S. Joshi

April 5, 2021

1 Introduction

The task is to make a parser-translator (that mean a complete syntax-directed translation scheme) for a game programming language. The game elements, its lexicon, and grammatical demands of its programming language are given here.

1.1 The Game

It is a 2048-game "family": Here, variations on the original 2048 game are to be also provided for. The variations are:

1. Allowing subtraction, multiplication and division in addition to the plain doubling operation at tile mergers. Thus, each move, when it is making two same-value tiles merge, may obliterate them together (making them 0 by subtraction), or reduce them to 1 by divition, or square them by multiplication. In this variation, the goal also will be flexible, any number not necessarily a power of 2 will be achievable.

2. Allowing variables in place of tile values to make puzzles: enabling questions like *how many operations it would take at the least, to double the maximum tile value in this position?*

Thus, the elementary operations of the 2048 game are to be provided, and little tweaks to them to allow all four arithmetic operations and variables are to be added.

1.1.1 The Operations

16 Moves: Add/Subtract/Multiply/Divide Left/Right/Up/Down. : Natural, except the extensions due to the airthmetic operations added, and the variables assigned.

Assignment: Assign $\ll$ *value* $\gg$ to $\ll x \gg$,$\ll y \gg$. : Setting a tile value.

Naming: Var $\ll$ *varname* $\gg$ is $\ll x \gg$,$\ll y \gg$. : Naming a tile. Each subsequent move will move the name also to the destination of this tile according to merging and stopping results. This may result in a tile getting several names.

Query: Value in $\ll x \gg$,$\ll y \gg$. : This value can be used in an assignment.

1.1.2 Semantics

This is a game programming language, so ultimately the semantics should result in the game states evolving and in their rendering on the screen. But, we want only a *translation* part to be implemented – it is about *compilation* only – therefore we settle for a text or array structure to be modified as a result of operations.

Moreover, we are not giving a full program structure yet. Thus, what you are supposed to make is an *interpreter*, like our command-line calculator program.

Thus, the developed program should begin an interactive shell. The beginning prompt is the start state, in which a random 2 or 4 tile is there, others are blank. The state may

be rendered like a list or a matrix or whatever is suitable. That part is not too important. Subsequently, each line of input from the user (who is really just a tester or evaluator like us instructors – neither a game programmer nor a player) should be interpreted into an action on the current state, and a new state should be produced and rendered as a response. Internally, keeping track of this current state must be done. Internally, variable name assignments to tiles also needs to be tracked. Whatever is needed to give a logically consistent interaction must be maintained in the internal data structures.

Thus, **the task is to make an interpreter for the language of interaction that allows the user give operation commands listed in 1.1.1.**

Example:

```
2048> Hi, I am the 2048-game Engine.
2048> The start state is:
-----------------
|   | 2 |   |   |
-----------------
|   |   |   |   |
-----------------
|   |   |   |   |
-----------------
|   |   |   |   |
-----------------
2048> Please type a command.
----> Hi, Good Morning!
2048> Sorry, I don't understand that.
----> Add Left
2048> Syntax error.
----> ADD LEFT
2048> You need to end a command with a full-stop.
----> ADD LEFT.
2048> Thanks, left move done, random tile added.
2048> The current state is:
-----------------
| 2 |   |   |   |
-----------------
|   |   | 2 |   |
-----------------
|   |   |   |   |
-----------------
|   |   |   |   |
-----------------
2048> Please type a command.
----> ASSIGN 7 TO 5,6.
2048> There is no tile like that. The tile co-ordinates must be in the range 1,2,3,4.
2048> Please type a command.
----> ASSIGN 7 TO 4,3.
2048> Thanks, assignment done.
2048> The current state is:
-----------------
| 2 |   |   |   |
-----------------
|   |   | 2 |   |
-----------------
|   |   | 7 |   |
-----------------
|   |   |   |   |
```

```
-----------------
2048> Please type a command.
----> 1,2 IS IS.
2048> No, a keyword cannot be a variable name.
----> 1,2 IS is.
2048> Thanks, naming done.
2048> Please type a command.
----> ...
...
```

1.2 Deliverables

Ultimately, you make a scanner, a parser-translator, and a main program that starts the interaction shell and continues to interact using the parser-translator. If you can incorporate the interaction loop into the translator itself (in the parser's rules-action section), fine, do so; but that will be too complicated. Easier is to have the main function implement the interactive shell, maintain the current state and do other internal bookkeeping, and call *yyparse* in response to each line, and rendering the output.

If you want to handcode everything, you are welcome, as long as you modularise the scanner, parser, and the interpreter shell. Please don't refurbish last semester's programs and submit them again. This is a **compiler construction** assignment.

Below is an example of how a typical font-end main module would look like (if implemented using `flex`, `bison`, etc.):

```
/* main.c */
...
/* extern tokenType yylex(...);: maybe needed only by the parser module. */
extern YYTYPE yyparse(...);
...
state CurrentState;
int main(void) {
  YYTYPE Response;
  CurrentState=Initialize();
  PrintInitialPrompt();
  RenderState(CurrentState);
  do {
    PrintPrompt();
    Response = yyparse();
    RenderState(CurrentState);
  } while(!eof(Response));
  return 0;
}
```

1.2.1 An Important Component for Evaluation

We want to automate the evaluation part. Therefore, apart from the usual interaction with the user (on the command prompt in a text terminal `stdout`),
provide on stderr the current state in a plain matrix given in a row-major sequence, and also variable names of each tile that is named after this matrix, following the same sequence (with no output for the unnamed tiles). The only separator for this output has to be the simple space '\40'. For each error only give -1 in the output line. For each new line in the main interaction, produce one single line on `stderr` here.

Thus, for the given example interaction above, the `stderr` output would be:

```
0␣2␣0␣0␣0␣0␣0␣0␣0␣0␣0␣0␣0␣0␣0␣0␣0
-1
-1
```

```
-1
2␣0␣0␣0␣0␣0␣2␣0␣0␣0␣0␣0␣0␣0␣0␣0
-1
2␣0␣0␣0␣0␣0␣2␣0␣0␣0␣7␣0␣0␣0␣0␣0
-1
2␣0␣0␣0␣0␣0␣2␣0␣0␣0␣0␣0␣0␣0␣0␣0␣1,2is
```

In this, ␣ is simple space `'\40'`. Remember, the state output comes only when there is some change either in the state or tile names. Tile names must come after the state each time, once assigned. Their positions will follow the moves, and accordingly this change should also be seen in this `stderr` output.

2 The Scanner

This was discussed in the class, tomorrow it will be copy-pasted here and updated.

3 The Parser

Now we know the **keywords** from 1.1.1 (they are **case-sensitive**:

```
ADD
SUBTRACT
MULTIPLY
DIVIDE
LEFT
RIGHT
UP
DOWN
ASSIGN
TO
VAR
IS
VALUE
IN
```

Make a suitable grammar to represent the operations in 1.1.1.

4 Deadlines and Evaluation

15th April is the deadline. Evaluation will be done by **automating interaction** with the interpreter executable on an Ubuntu 20.04. That means test cases of input lines will be given and the output tested, therefore, pay attention to the point 1.2.1 about the `stderr` output.

What to submit? An archive (`zip, tar.gz`, etc.) that has everything bundled, including a make file. Assume that we have `bison, flex, gcc`.

`Hope you enjoy doing this assignment.`