DPL

(P3)

Yeray Méndez Romero

[yeray.mendez@udc.es](mailto:yeray.mendez@udc.es)

Daniel Rivera López

[d.rivera1@udc.es](mailto:d.rivera1@udc.es)

Index:

[Changes to the original code: 3](#_Toc501121395)

[The top level: 6](#_Toc501121396)

[-User manual: 6](#_Toc501121397)

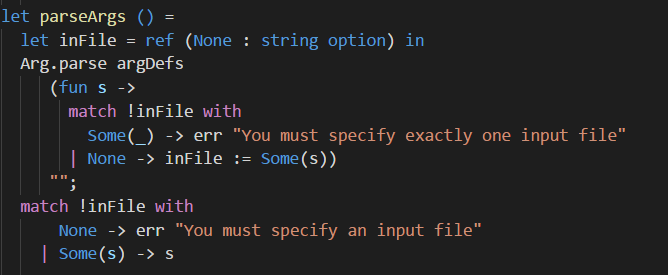
[-Execution examples: 7](#_Toc501121398)

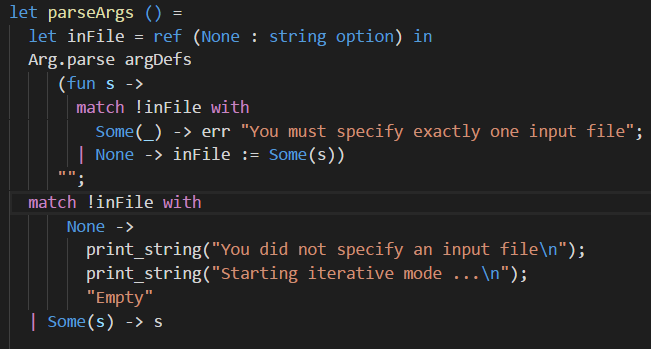
# Changes to the original code:

In order to build the top level, we needed to make some changes to the original implementation:

1. Changes to ‘parseArgs’:

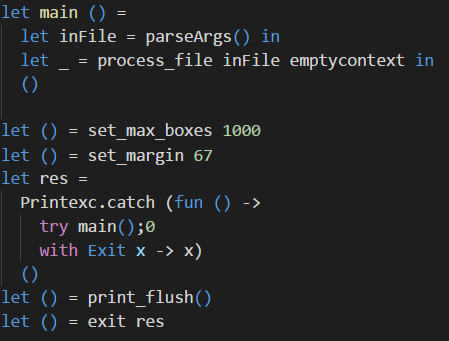
The error that happened when no input was provided has been changed to return a value ‘Empty’ that will be later used to start the top level.

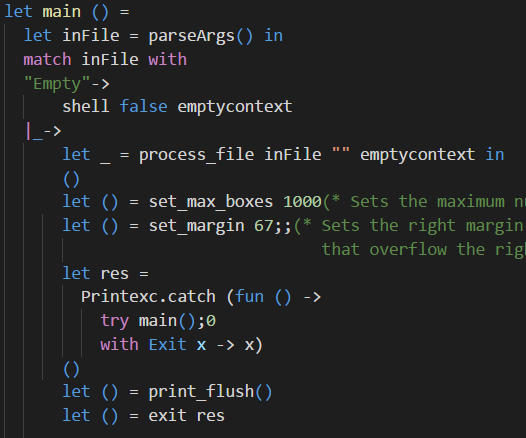




1. Changes to ‘main’:

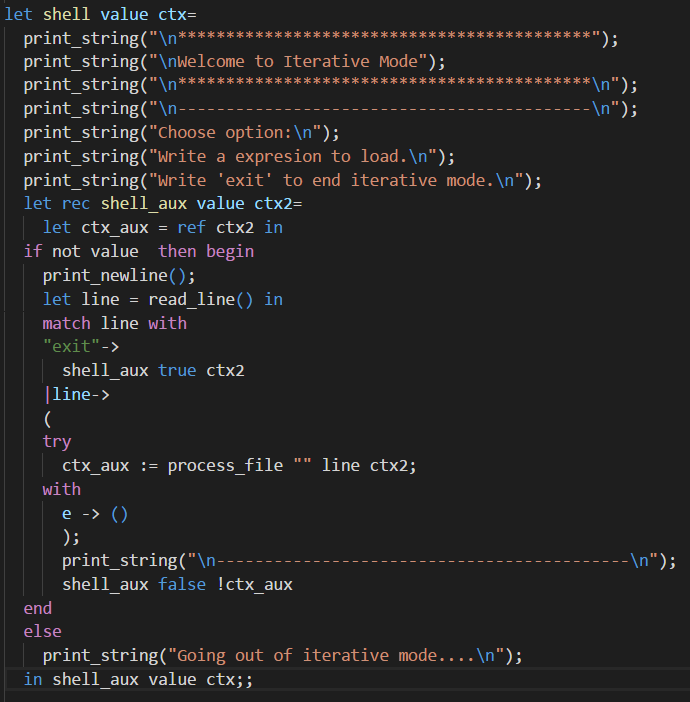
Now the ‘inFile’ in ‘main’ is matched to detect “Empty” and start the top level.





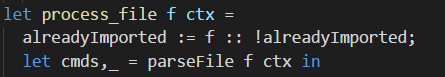
1. New function ‘shell’ added:

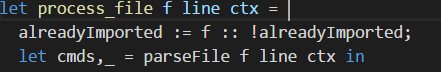
This new function starts the top level and process each line wrote by the user.



1. The function ‘process\_file’ now receives a new parameter:

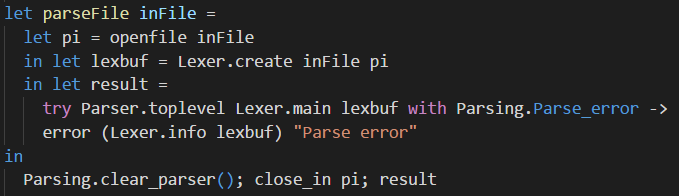
This new parameter ‘line’ represents each line that the user writes on the top level.

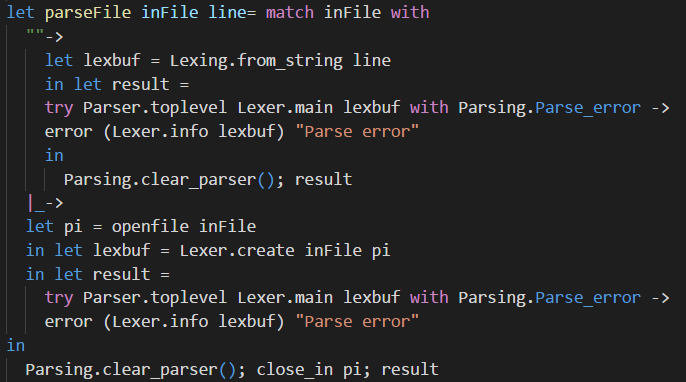




1. Changes to ‘parseFile’:

Now receives the new parameter ‘line’ and use it when the ‘inFile’ does not contain anything to build a ‘lexbuf’ with it and send it to the Parser.





# The top level:

## -User manual:

To start the top level just execute the program with no arguments.

To end the top level write ‘exit’ on the standard input and press Enter.

Inside the top level you can write lambda expressions on the standard input and press Enter to make the program process them.

The reserved words are:

(\* Words \*)

"if", "then", "else", "true", "false", "lambda" ,"timesfloat”, "succ", "pred", "iszero", "let", "in".

Aside the usual ones, here is an explanation on their use:

- “lambda”: defines the start of a function.

- “timesfloat”: function that multiplies two float numbers. \*1

- “succ”: function that calculates the next number of the given one. \*1

- “pred”: function that returns the previous number of the given one. \*1

- “iszero”: function that checks if the value given is zero or not.

- “let” and “in”: allow to define bindings in only an expression.

Example: let a = 1 in a; output: 1

\*1 : These functions can be called with anything that the parser allows, but if you do that the program won’t be able to evaluate them and will return exactly the input.

(\* Symbols \*)

There is a lot of reserved symbols too, but here will only be shown the basic ones, for more details check the files “lexer.mll” and for all the possible ways to combine them check “parser.mly”.

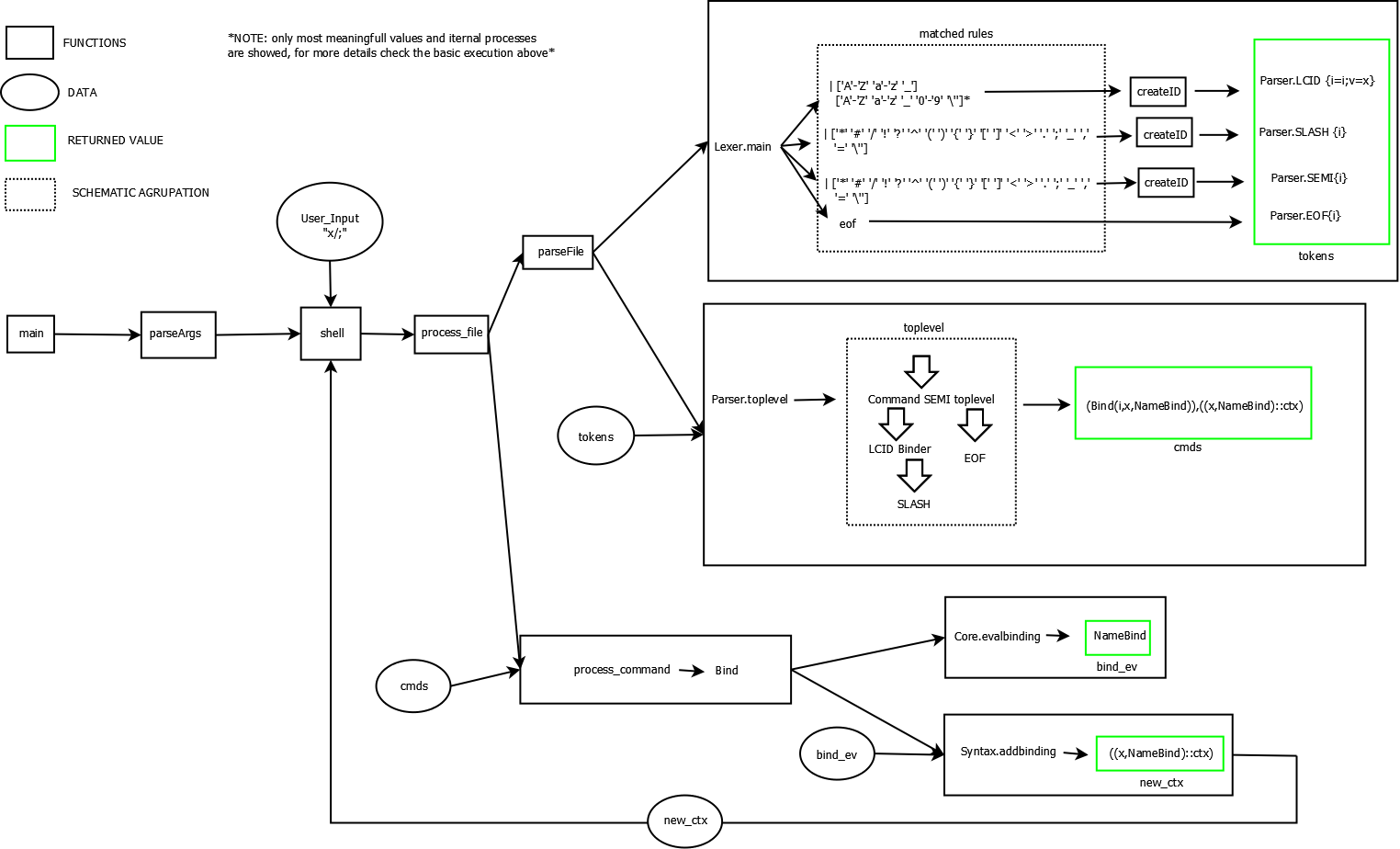
* “;”: each command must end with one.
* “/”: used when declaring a variable without value
* All usual lambda calculous symbols will work as expected.

## -Execution examples:

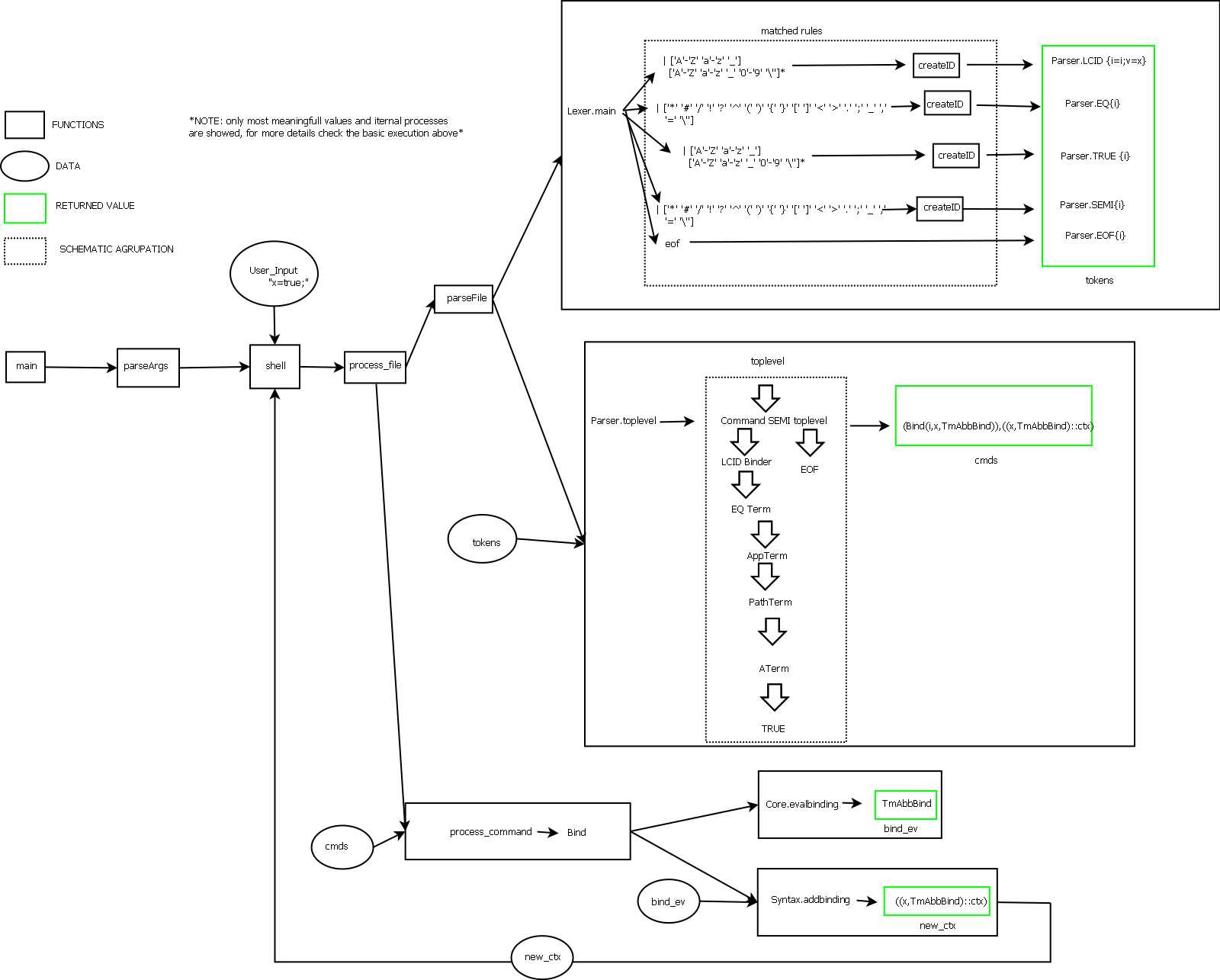
-> Basic execution:

The program starts on the “main” function and calls “parseArgs”, because the program has been launched without parameters “parseArgs” returns ‘Empty’ witch indicates to the “main” function to start the top level by calling the function “shell”. The function “shell” receives the user expression from the standard input and calls “process\_file” with the user expression, “process\_file” calls to “parseFile” where the “Lexer” is called and the user input is converted into tokens that return to “parseFile” to be the input in the “Parser” call. In the “Parser” the tokens will be compared with the grammar and a list of commands in addition to a context list will be returned. At this point the program flow returns to “process\_file” and calls “process\_command” with each command that returns from “parseFile”. Inside “process\_command” Binding and Evaluation commands are evaluated, the result from the evaluation is printed to the user and a new context is returned all the way to the “shell” function where will be stored for its use with the next user input.

-> Input: x/;



-> Input: x = true;



-> Input: x; (after executing x/;)

