Lettuce Wrap: An Interactive Language Interpreter

Reece Suchocki, Spencer Wilson

April 19, 2023

1 Status Summary

As a semester project, our team will build an interactive language interpreter coded in Scala. Users of the system can author programs in our domain-specific language (DSL) and view the execution steps of that program in a step-wise fashion. The inspiration for our project comes from the Lettuce Playground Project developed at CU Boulder in CSCI 3155 Principles of Programming Languages. This tool will go further to allow the client to change language features, such as dynamic or static scope, short-circuiting, lazy evaluation, and implicit type conversions while still viewing the program evaluation order.

1.1 Work Done

Written description of the work done in the first week of your project and (in the case of multi-person teams) the breakdown of work across team members.

Reece Suchocki
Initialized the Lettuce Wrap GitHub repository, *hyperlinked here*, and installed dependencies: Material-UI, our boilerplate React application, and the Scala Play back-end. Reece then connected the initial endpoints that POST data between the React front-end components and the Scala back-end. Reece then created the UserInput, Expression, and NewExpression components (UI) to handle step-through evaluation based on the user input.

Spencer Wilson
Spencer expanded the Lettuce playground parser to the defined grammar in submission 5 (Ref. Index 4.1). Then created an small step interpreter to work for any collection of evaluation conditions of lazy-eager evaluation, type casting implicitly or not, static vs dynamic scope. This was performed as a test integrated process to build an expanding suite of 50+ unit tests. Spencer went on to do the initial connection of the back-end to front-end. In doing this Spencer added additional classes ‘UserInput‘ and ‘EvaluationResponse‘ to encapsulate the logic of parsing user input json and construct the json response for the front-end.

e.g. UserInput

```
{
    evaluationConditions: {
        scope: <>,
        types: <>,
        lazyEager: <>,
    },
    expression: <>
}
```

e.g. EvaluationResponse

```
{
    "expression": "<>",
    "value": "<>",
    "steps": ["<>"]
}
```

Reece Suchocki & Spencer Wilson

Finally, Reece and Spencer collaborated to complete the current front-end, to demonstrate step-through capability of the small-step interpreter for the end user. Please note that this parlleled development effort would not have been possible without the detailed design work completed collaboratively during project 5.

1.2 Changes & Issues

Has anything changed so far in your approach to the project from the initial design in Project 5?

Front-end

Challenges included learning the React and Material-UI syntax and data flow, as well as communication with the Scala back-end. Our first issue encountered was with the CORS (Cross-Origin Resource Sharing) policy. It is a security issue to have localhost:3000 and localhost:9000 communicate. The issue took an afternoon to resolve modifications to the package.json file. The second issue encountered with the front-end was in trivial variable naming. The back-end was passing through ‘data.message‘, which took us some time to identify as inconsistent of what the front-end was expecting.

Additionally, the front-end UI has developed further for ease of access of the user (Ref. Index 4.2). We have adopted a vertical design layout to accommodate for longer expression steps, which might be harder to visualize in the horizontal design layout of our sketches.

Back-end
Challenges in determining how to implement static scope in small step. Ultimately achieved this by using closures. Curious about this solution as as closures are actually used to implement static scope in a big step interpreter.

In integration to the front-end, we have lost the ability to execute the unit tests for the program, which was a real bummer, but this has since been resolved.

Concerns about how to obfuscate the code enough that students in CSCI 3155 cannot easily use this open-source code as a copy-and-paste repository for their own assignments. We got around this using the continuation passing style (CPS) and some patterns discussed in the next section of this document.

We are still concerned about testing the languages as, we have effectively built 8 separate language semantics for the same syntax and it's not reasonable, in time constraints, to test all of them. We have some reasonable test coverage and currently support integration tests across the possible semantics.

1.3 Patterns

Now that you have more of your system implemented, please describe the use of design patterns so far in your prototype and how they are helping you or your design.

1. MVC-ception: React.js has it's own MVC pattern in the framework. We also separate our code into an MVC architecture.
2. Composite pattern for the expr data structure.
3. Interpreter pattern integrated to the composite pattern for expression evaluation.
4. Template pattern implemented to integrate with the step-function of the interpreter to encapsulate variance in language semantics to relevant evaluation condition as needed while leaving core logic of the interpreter in-situ where possible.

2 Next Iteration...

Provide an estimate of how much more work needs to be done for your team to have implemented the design that you presented in Project 5 (with any design changes that may have occurred). What are your plans for the final iteration to get to the Project 7 delivery? What do you plan to have

done by 5/3 when the overall project is due?

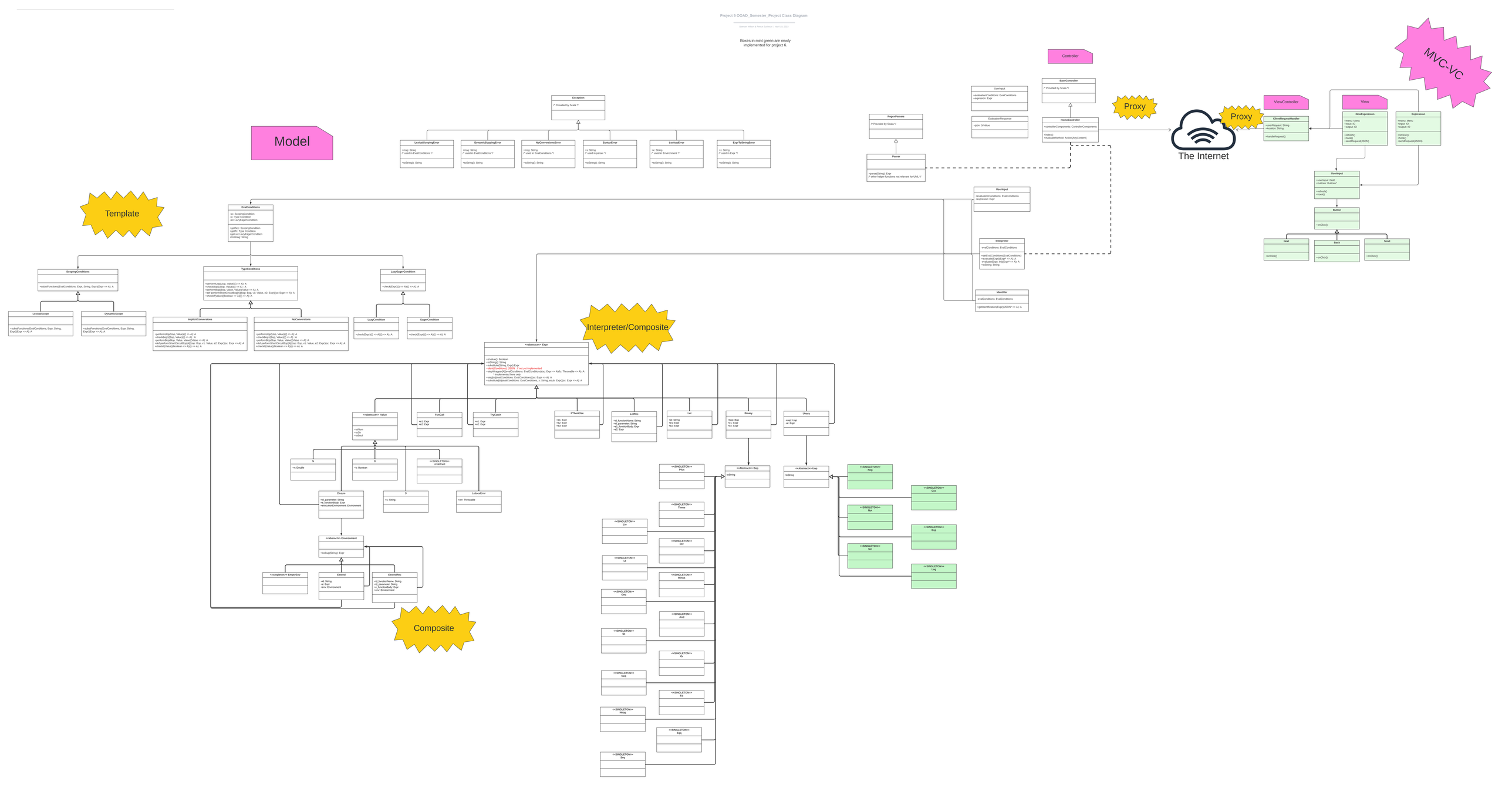
Currently, we do not support unary operations, but this is under construction as of time of this submission (3pts). Currently we do not support the ability for the user to change the language semantics via the GUI, but we intend to implement this (5pts). Currently we do not support language syntax highlighting to better demonstrate the reason for langauge reductions, while we will explore this and hope to implement this by end of term, we are not certain if it is reasonable to accomplish in scope (13pts). On a 0, 1, 2, 3, 5, 8, 13 point agile scale for a two week development cycle noting that the development team is only part time on the project. We did not track capacity during spring 5 and sprint 6.

3 Class Diagram

Project 5 OOAD_Semester_Project Class Diagram

Spencer Wilson & Reece Suchocki | April 18, 2023

Boxes in mint green are newly implemented for project 6.



4 Index

4.1 Lettuce Wrap Grammar [1]

$e \Rightarrow$	v	$uop \Rightarrow$	$-$
$\mid$	$e_1(e_2)$	$\mid$	!
$\mid$	try { e_1 } catch { e_2 }	$\mid$	sin
$\mid$	if (e_1) { e_2 } else { e_3 }	$\mid$	cos
$\mid$	let x = e_1 in e_2	$\mid$	exp
$\mid$	let rec f = function(x) e_1 in e_2	$\mid$	log
$\mid$	e_1 *bop* e_2		
$\mid$	*uop*(e_1)		

bop $\Rightarrow$	+	$v \Rightarrow$	< number >
$\mid$	*	$\mid$	< boolean >
$\mid$	/	$\mid$	undefined
$\mid$	$-$	$\mid$	< string >
$\mid$	&&	$\mid$	ERROR
$\mid$	$\mid\mid$	$\mid$	function(x) e
$\mid$	==	x	`is an identifier`
$\mid$	===		
$\mid$	! =		
$\mid$	! ==		
$\mid$	<		
$\mid$	$\leq$		
$\mid$	>		
$\mid$	$\geq$		
$\mid$	;		

[1] Please note that the above grammar is in Backus–Naur form for concrete sentences of the language.

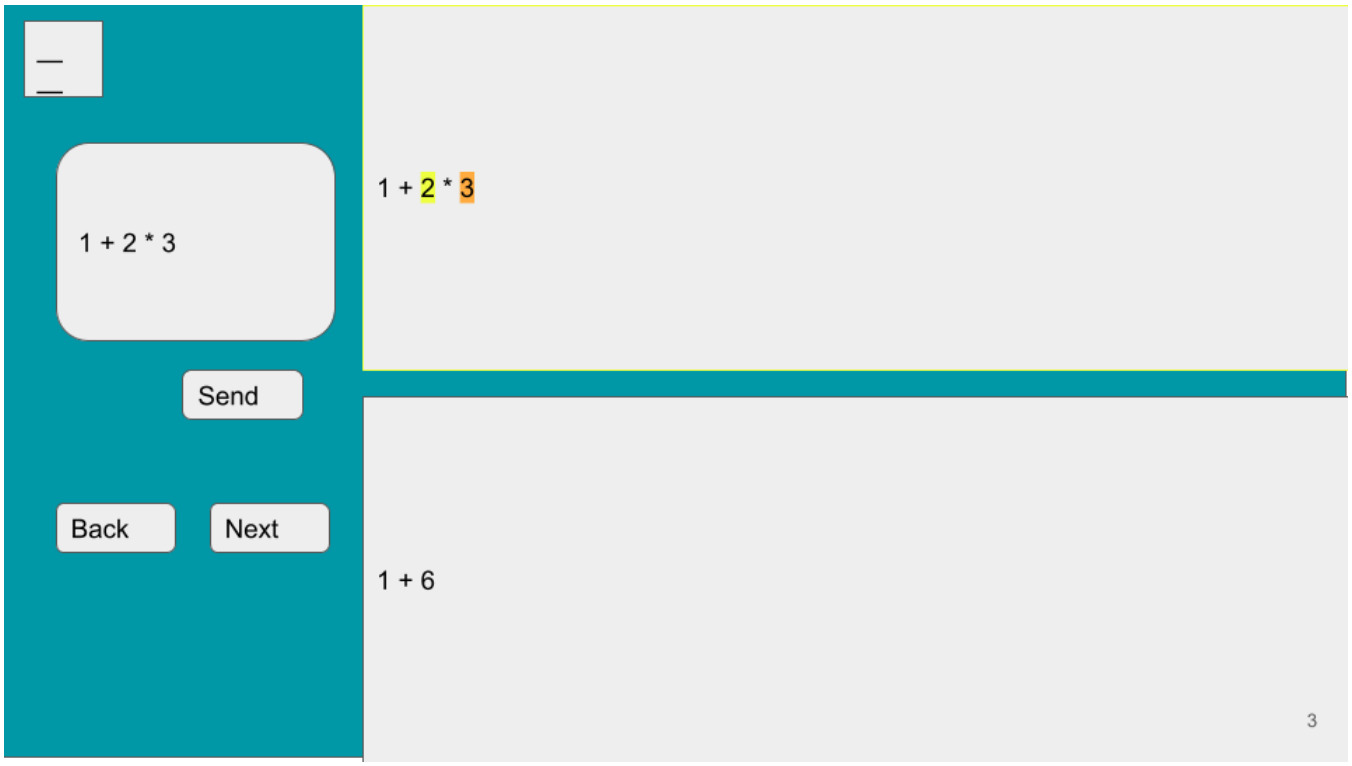


Figure 1: Early design sketch of the Lettuce Wrap UI

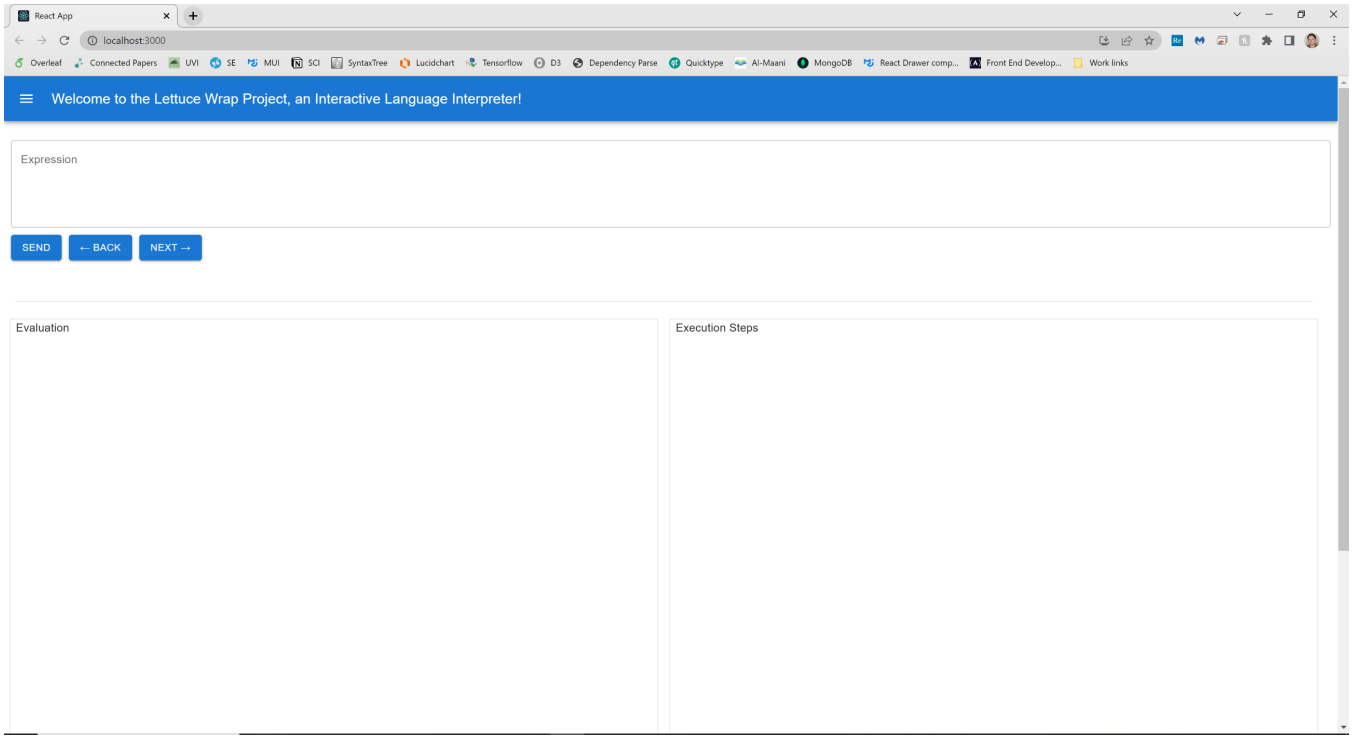


Figure 2: Iterated UI of the Lettuce Wrap project