README

What the artifact does

The purpose is to illustrate the usefulness of our proposed framework which can support bidirectional live programming for incomplete programs. The artifact supports developers not only to write incomplete programs (using a simple functional language) and observe the output (e.g., a web page) with hole closures in editor services, but also to directly manipulate them to synchronize the program with the output. (You can refer to the small example in the last section.)

Where it can be obtained

Docker Image for Demonstration

https://drive.google.com/file/d/1G7Q6lNxsd8rmOkMnERqaUy6fVqJwy9oi/view?usp=sharing

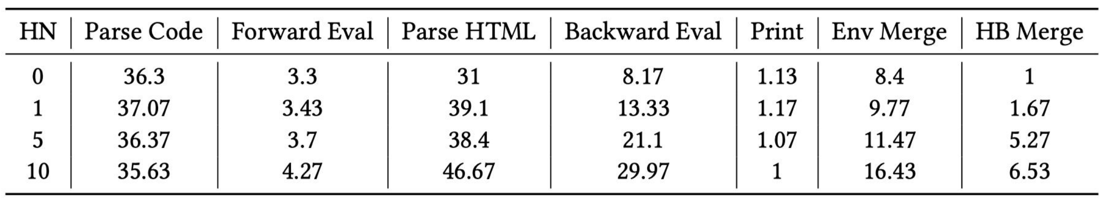
Open Source Library

https://github.com/1BidirectionalPreview/BidirectionalPreview

How to repeat/replicate/reproduce the results presented in the paper

In our paper, we demonstrate the usefulness of the system through three examples, and demonstrate the effectiveness through experiments on statistical running time. The experimental results are as follows.

Table 1: Running Time of Bidirectional Evaluations



HN	Parse Code	Forward Eval	Parse HTML	Backward Eval	Print	Env Merge	HB Merge
0	36.3	3.3	31	8.17	1.13	8.4	1
1	37.07	3.43	39.1	13.33	1.17	9.77	1.67
5	36.37	3.7	38.4	21.1	1.07	11.47	5.27
10	35.63	4.27	46.67	29.97	1	16.43	6.53

HN: Number of holes in program; Env Merge: Environment Merge; HB Merge: Hole Bindings Merge

For implementing examples in the paper

There are three examples in the paper, the initial incomplete code of which can be found in examples folder(html03_classmates_book, ex02_quick_sort, and ex01_student_grades) in the source code. (For details, see Section 2 Overview and Section 5 Two Applications in our paper.)

a) Copy the incomplete program into the code editor, then click the "Eval" button to run, and get the incomplete output in the Output window and hole closures in the Context window.

b) Modify the output or hole closure (click hole values to jump to its closure) to the expected result, then click "Uneval and Update" to get the automatically updated program.

c) Click the "Eval" button again to see if the updated program runs with modified output and hole closures.

For repeating the experiments in the paper

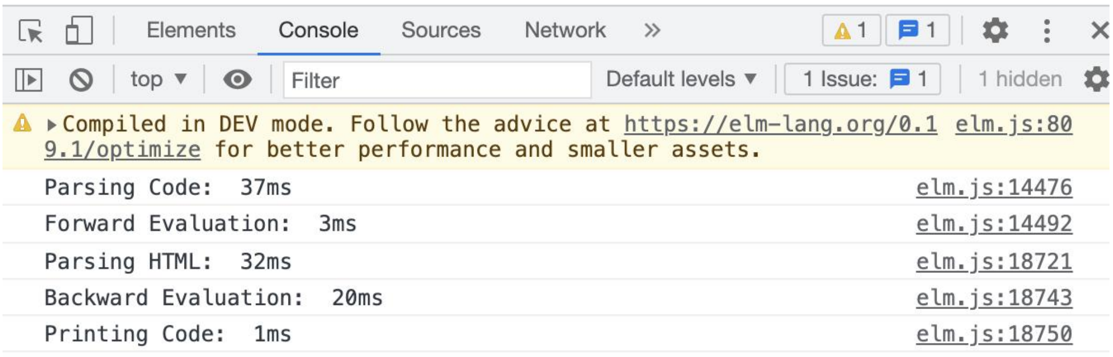
Our method is to add timing statements to the seven parts of Parsing Code, Forward Evaluation, Parsing HTML, Backward Evaluation, Printing Code in the compiled elm.js file. (Two detailed tests, Environment Merge and Hole Bindings Merge, are omitted here, but they are not technically difficult.) For the convenience of reproduction, we have modified the elm.js file to the elm-test.js file. Just change the elm.js file linked on line 8 to the elm-test.js file when running the index.html file using vim installed. The timing statement is as follows.

```
var startParseCode = new Date().getTime();
var parseResult = $author$project$Parser_$parse(code);
var endParseCode = new Date().getTime();
console.log('Parsing Code: ', `${endParseCode - startParseCode}ms`);
```

a) Copy the code from line 87 to line 162 in *html01_table_of_states* file from the Examples folder of the source code into the code editor.

b) Click the “Eval” button to get running time for Parsing

Code, Forward Evaluation, which is recorded in the console of Browser Developer Tools. Then, Click the "Uneval and Update" button to get running time for Parsing HTML, Backward Evaluation, Printing Code.



```
Compiled in DEV mode. Follow the advice at https://elm-lang.org/0.1   elm.js:80
9.1/optimize for better performance and smaller assets.
Parsing Code:  37ms                                                elm.js:14476
Forward Evaluation:  3ms                                           elm.js:14492
Parsing HTML:  32ms                                                elm.js:18721
Backward Evaluation:  20ms                                         elm.js:18743
Printing Code:  1ms                                                elm.js:18750
```

c) Incrementally increase the number of holes in the *states* list (line 127) from 1 to 10. For example, change the first item (i.e., `["Alabama", "AL?", ""]`) to a hole (i.e., `_`), and leave the rest unchanged. After performing operations in b), change the second item to a hole as well and still leave the rest unchanged. Until 10 holes are added to the *states* list. Operations need to be performed ten times for each number of holes. The final result is the average of ten runs. Note that it needs to be refreshed and run again after each run, otherwise the time will be inaccurate due to the existence of the cache.

How to install it

There are two ways to install our artifact: one way is to configure the environment directly on localhost; the other way is to use docker.

Install Directly

1. Install Elm

https://guide.elm-lang.org/install/elm.html

2. Download Source Code

https://github.com/1BidirectionalPreview/BidirectionalPreview

3. `cd BidirectionalPreview` (The root directory containing the *src* folder)

4. Compile Project: `elm make src/Main.elm ––output=elm.js`

5. Open index.html using browser

Install Using Docker

1. Download the docker image file (bidirectional_preview.tar) from the artifact link

2. Import the image and name it *bidirecitonalpreview*

```
docker import bidirectional_preview.tar bidirectionalpreview:latest
```

3. Check if import successfully

```
docker images
```

4. Create a container with port mapping and name it BiPreview.

Port 8000 of the container is mapped to port 8080 of the host.

```
docker run -ti -d --name BiPreview -p 8080:8000 -it
 bidirectionalpreview /bin/bash
```

5. Start the container BiPreview

6. Enter project:

```
cd /root/BidirectionalPreview
```

7. Compile project:

```
elm make src/Main.elm --output=elm.js
```

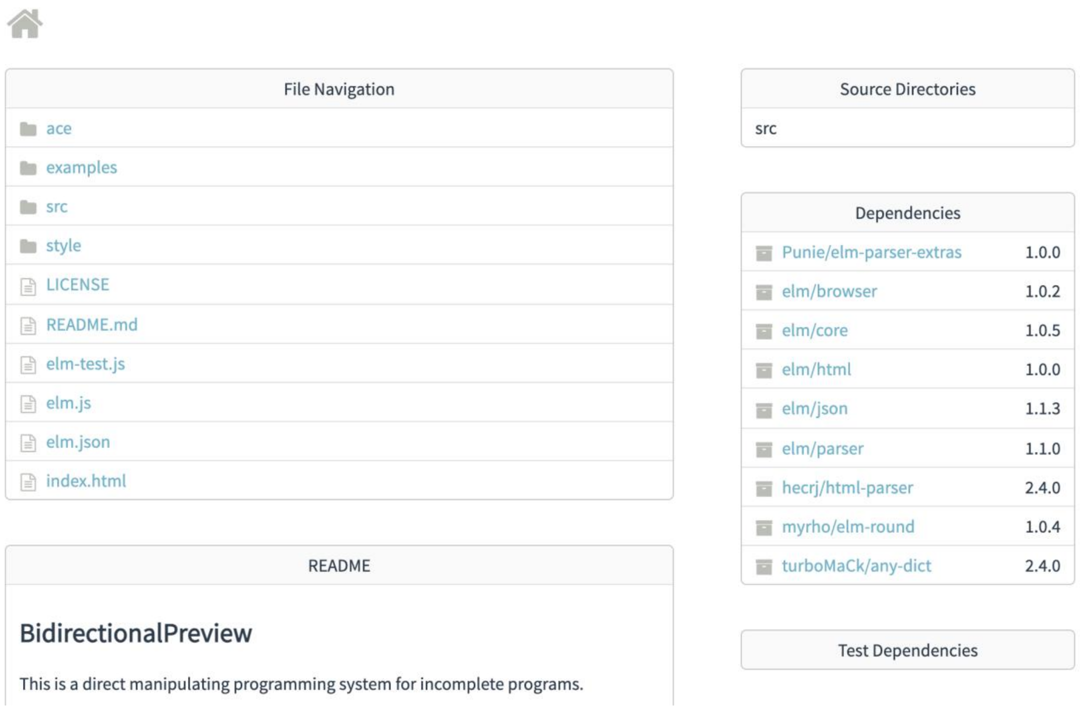
8. Starts a server at: http://localhost:8000

```
Elm reactor
```

9. In the host, visit url:

http://localhost:8080

10. Click index.html in the web page you just opened



File Navigation

- ace
- examples
- src
- style
- LICENSE
- README.md
- elm-test.js
- elm.js
- elm.json
- index.html

README

BidirectionalPreview

This is a direct manipulating programming system for incomplete programs.

Source Directories

src

Dependencies	
Punie/elm-parser-extras	1.0.0
elm/browser	1.0.2
elm/core	1.0.5
elm/html	1.0.0
elm/json	1.1.3
elm/parser	1.1.0
hecrj/html-parser	2.4.0
myrho/elm-round	1.0.4
turboMaCk/any-dict	2.4.0

Test Dependencies

How to use it

Initial Incomplete Programs

1. In the editor on the left, write a program based on the syntax mentioned in the paper.
    - Hole: `_` (underscore)
    - Lambda Expression: `\x=>x+1`
    - Function Call: `(\x=>x) 3`
    - List Construction: `1::2::[]` (or, `[1,2]`)
    - Tuple: `(1,2)`
    - Let-Binding: `let a = 1 in a`
    - Letrec-Binding: `letrec f = ... in ...`
    - If Conditional: `If x==y then ... else ...`
    - Case Expression: `case x==y of true=>...|false=>...|...`
2. Click the "Eval" Button to see the output

Direct Manipulate on Output/Context

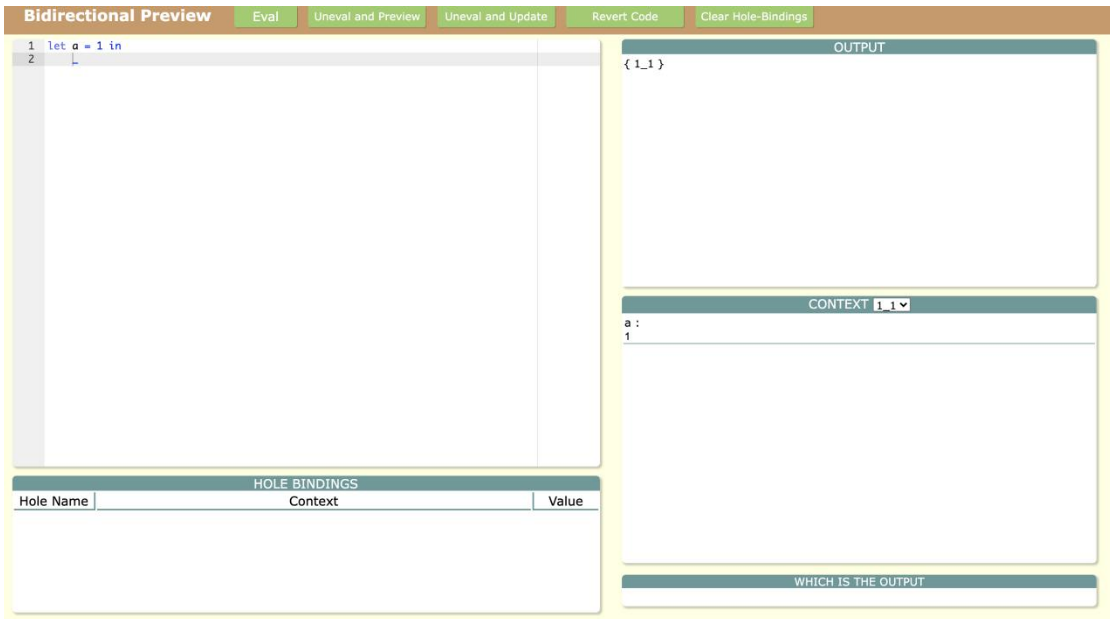
1. Modify the text in the Output window
2. Modify the values in the Context window
    a) Click hole values to jump to its closure
3. Click “Uneval and Preview” button
    a) To see the preview of the code changes
    b) Click the “Revert Code” button to revert to the original code
4. Click “Uneval and Update” button

a) Make sure the updated code and cannot retract

A Small Example

We use this small example to show: for incomplete programs, our tool allows developers to modify the context of code blanks, thereby modifying the corresponding program, so that the program can run again to get the modified context.

1. Write a simple incomplete program `let a = 1 in _` in the editor, which means declaring a variable ***a*** with a value of **1** and the rest of the code is a temporarily blank indicated by an underscore. Then click the "Eval" button. The result is as shown below. **{1_1}** in the Output window meas the incomplete output of the incomplete program. The context of the code empty in the program is shown in the Context window, where display the variable ***a*** and its value.

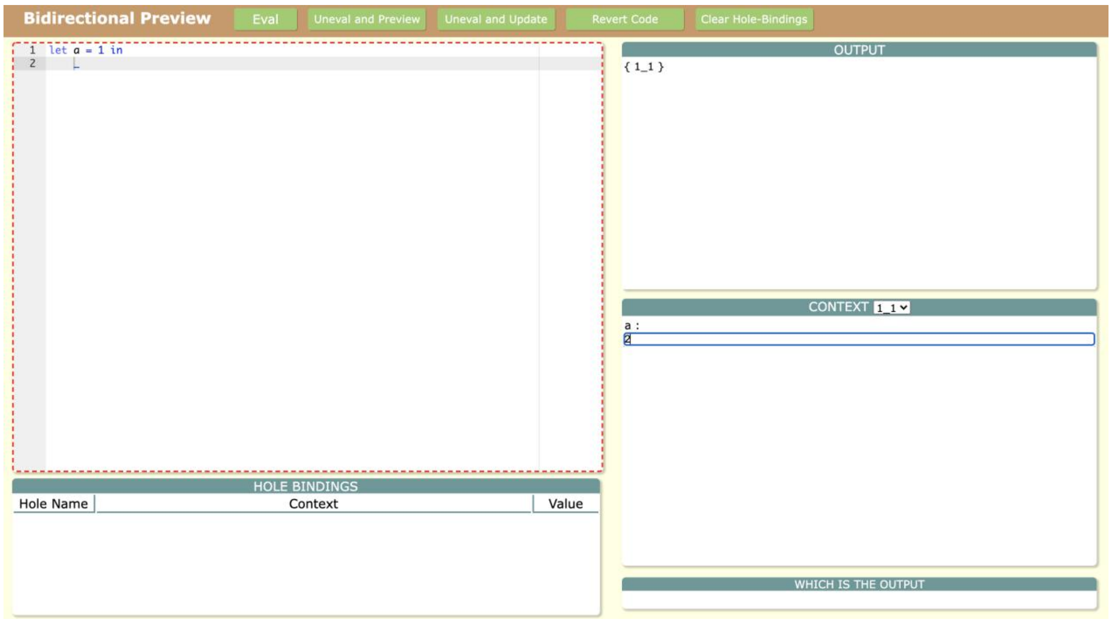


```
let a = 1 in
```

```
{ 1_1 }
```

```
a :
1
```

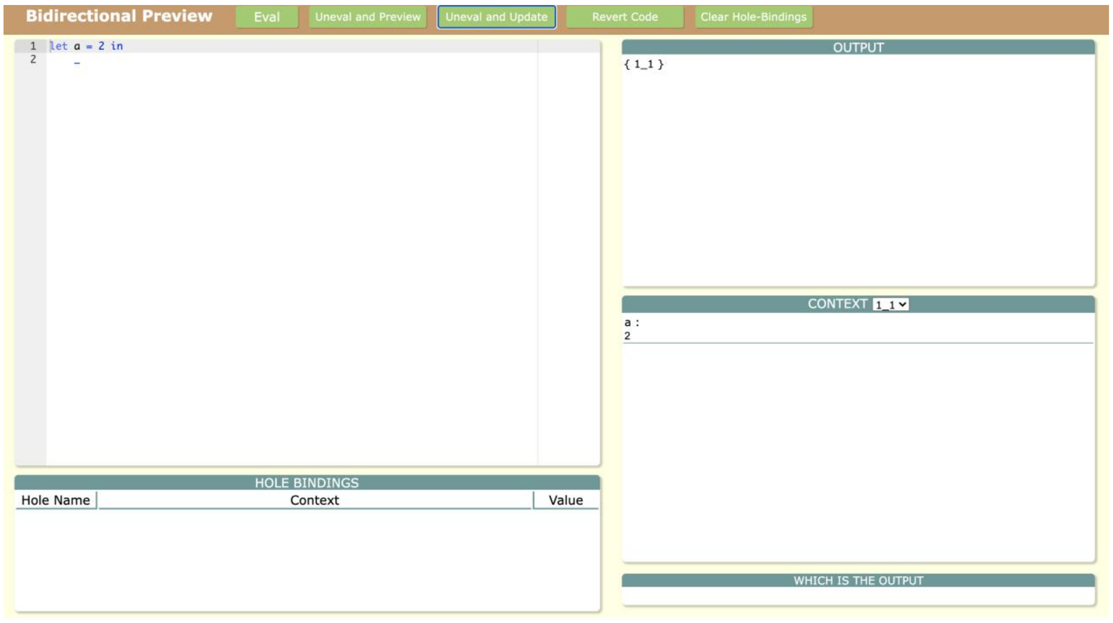
2. Modify the value of variable ***a*** in the Context window to 2 as show in the first figure. Then, click the "Uneval and Update" button. The result is shown in the second figure.



```
let a = 1 in
```

```
{ 1_1 }
```

```
a :
2
```



```
let a = 2 in
    _
```

```
{ 1_1 }
```

```
a :
2
```

Hole Name	Context	Value

3. Click the "Eval" button. Then, check if the value of the variable ***a*** in the Context window is updated to 2.