CS4215

TEAM PONTEVEDRA

Source to EVM Compiler

Authors:
Yuchen WANG (A0200093X)
YiJia CHEN (A0200138X)

NUS School of Computing

April 14, 2022

1 User-level Documentation

The Source-to-EVM current supports the compilation of a subset of the Source to bytecode for the Etherum Virtual Machine. Syntax follows that of Source.

1.1 Supported Features

- Integer arithmetic
- Boolean operations
- Declaration of variables and constants
- Functions
  - Function declarations and applications
  - Named functions and anonymous functions
  - Nested functions
  - Recursive functions
  - Tail recursive functions
- Conditionals
  - Ternary operator
  - If-else statements
- While and for loops
  - Note that `break` and `continue` are not supported

1.2 Operators

The following operators are supported:

Operator	Type of operand 1	Type of operand 2	Return type
+	Number	Number	Number
-	Number	Number	Number
*	Number	Number	Number
/	Number	Number	Number
===	Number/Boolean	Number/Boolean	Boolean
<	Number	Number	Boolean
<=	Number	Number	Boolean
>	Number	Number	Boolean
>=	Number	Number	Boolean
&&	Boolean	Boolean	Boolean
||	Boolean	Boolean	Boolean
!	Boolean	-	Boolean

The ternary conditional operator is also supported:

```
condition ? if-true : if-false
```

1.3 Things to note

1.3.1 Output

Similar to the Source interpreter, the compiled code will always return the result of the last statement of the given program that has return results. However, unlike the Source interpreter, if none of the statements in the program has return results, the code will return 0.

Examples:

```
3+4;
2+1;
// output 3


3+4;
let x = 2+1;
// output 7

let y = 7;
let x = 2+1;
// output 0
```

1.3.2 Unused return results

Due to the compiler's reliance on the EVM's stack for exiting from functions, **any** unused return values from any statements **within functions** will cause undefined behaviour, and will likely lead to EVM errors. Such statements can still be used outside of functions.

Examples:

```
function f() {
3 + 4; // return value 7 not used, will cause EVM error
return 8;
}

function f() {
3 + 4; // return value 7 not used, will not cause EVM error but should still
be avoided
}

function f() {
let x = 3 + 4; // return value 7 is used in assignment
return 8;
}

3 + 4; // this is fine as it is not in a function
function f() { return 8; }
```

1.4 Compile-time Checking

There is no static type checking in the compiler, but the compiler will check and throw exceptions for the following:

- Reassigning values to constants

  e.g.
  ```
  const x = 2;
  x = 3; //reassigning const, compiler will throw exception here
  ```

- Referring to undeclared name

  e.g.
  ```
  const x = 2;
  y + 4; //y not declared, compiler will throw exception here
  ```

- Using an unknown operator

  e.g.
  ```
  1 $ 2; //$ is not a supported operator
  ```

Note that there will not be any line number information in the error messages.

2 Developer-level Documentation

In order to define the relation $\cdot[\cdot \leftarrow \cdot]\cdot$ we employ as usual an inductive definition using the following rules.

2.1 Inductive Definitions

$$\frac{}{v[v \leftarrow E_1]E_1} \text{ for any name } v \qquad \frac{}{x[v \leftarrow E_1]x} \text{ for any name } x \neq v$$

$$\frac{E_1[v \leftarrow E]E_1' \qquad E_2[v \leftarrow E]E_2'}{E(E_2)[v \leftarrow E]E_1'(E_2')}$$

2.2 Garbage Collection

Garbage collection is implicitly implemented in our compiler design.

Testament:

- For loop that calls a function that creates a bunch of variables. Each function call creates a new Environment, but the heap (memory) size will stay relatively constant

```
function f() {
    let x = 5;
    let y = 6;
}
for (let i = 0; i < 3; i = i + 1) {
    f();
}
```

- tail_recursion would take up constant space in the heap.

2.3 Compiler functions

2.3.1 `compile_expression`

2.3.2 `compile_sequence`

2.3.3 `compile_constant`

2.3.4 `compile_conditional`

2.3.5 `compile_while_loop`

2.3.6 `compile_for_loop`

2.3.7 `compile_lambda_expression`

2.3.8 `compile_application`

2.3.9 `compile_tail_call_recursion`

2.4 Running the suite

A Appendix