The Towel Report

Zihang Chen (zc2324) Baochan Zheng (bz2269) Guanlin Chen (gc2666)

December 21, 2015


DON'T
PANIC

Contents

Chapter 1

Language Overview

Towel is a stack-based, general-purpose, postfix-syntaxed, dynamically strong-typed, functional, language targeted at the Towel Virtual Machine.

1.1 Stack-based[1]

Although stack-based programs are hard to reason about, it's really efficient and powerful to use once you get the gist of it. It is comes with natural support for imperative programming paradigm. It also enforces programmer's to think about the evaluation order of their potentially side-effected code.

1.2 General-purpose

Towel is designed with the universe in mind. So the syntax of it is expressive and reasonable with as few limitations as possible, yet easy to understand.

1.3 Postfix-syntaxed

The relationship between postfix syntax and stacks is just like a towel to a hitchhiker, so why not?

1.4 Dynamically Strong-typed

This means that Towel maintains types at runtime with no implicit type castings whatsoever, the compiler does not do type checking. However, it does scope analysis for every piece of Towel code.

1.5 Functional

Once you have full-fledged functional framework, you have the whole world. For example, Church booleans, Church numbers, etc. Or more realistic, you can use functions to emulate records, or even object-oriented programming!

1.6 Towel Virtual Machine

Why a virtual machine? Because we think this is the most portable and easy way to do a prototype for a new language. After you got the language working, you can do all kinds of crazing things, like compiling the bytecode that the VM reads into a native application.

[1] It's not only stack-based, it's actually stack-stack-based. See also section 3.1.

Chapter 2

A Short Tutorial to the Towel Programming Language

Welcome to the tutorial[1] and don't panic!

Let's implement a summation function!

Example

```
import 'std' @ (1)

bind (2) Fold-left ,\ (3) Acc Xs Fun,
  (Xs ?# (4) ift (5) Acc,
     (Acc Xs #hd Fun (6) Xs #tl Fun` (7) Fold-left@ (8)))
also Sum (0 (+` Fold-left) (9) /flip (10))
then ([1 10 11 20] (11) Sum !println (12))
```

Now, let me explain something:

1. By default, this isn't any standard library function available until you explicitly import the module `Std`. See also subsection 3.10.1.
2. Bind some name (see also section 1.3) to a value. In this case we are binding a function (represented by `,\`) to the name `Fold-left`! For more on binding values to names, see also section 3.9
3. The token `,\` is just a token `fun` with a backquote. More on backquotes and function forms, please see section 3.6 and section 3.8.
4. Test if `Xs` is an empty list using the standard library function `?#`. For more on the standard library, see chapter 4.
5. This shows you how to do conditional branching in Towel. There are up to 10 kinds of `if` forms in Towel, each kind of them tests the TOS to see if it agrees with the predicate. See also section 3.7.
6. Calling the function `Fun` which is passed as an argument! Functions are first-class citizens in Towel.
7. Backquoting the name `Fun` to prevent it from executing. Because in this case, we want this function to simply be a value that can be passed as an argument.
8. Tail calling the function bound to `Fold-left` by appending an `@` symbol to the name.
9. Partial function application here. We are applying only the addition function to the trinary function `Fold-left`, to get a binary function that adds each element of the list to the accumulator. The two arguments of this function is the list and the initial value of the accumulator.

[1] We will assume you know the basics about how a stack-based language works.

10. Then, we use the standard library function `/flip` to flip the rest two arguments of the partial-applied `Fold-left` so that it accepts the initial value then the list. After this, we apply a zero to the flipped partial-applied `Fold-left`, so that we obtain a unary function which does the same as the last function we get, except it accumulates from **0**.

    The `also` clause of the `bind-then` form binds the name `Sum` to this unary function we just obtained.

11. List literal. See also subsection 3.4.3 for details on list literals.

12. Calling the `Sum` function in the `then` clause of the `bind-then` form to sum the list we just created, and use standard library function `!println` to print the Answer **42**!

To see more examples, please read chapter 6. If you are interested, and want to learn more about this novel language, do read on for the Towel Reference Manual, where you will learn every aspect of the language.

Part I

The Towel Reference Manual

Chapter 1

Lexical Elements

1.1 Keywords

Keywords in the Towel programming language are defined as follows:

Lexeme Definition

```
IFGEZ ::= "if>=0"
IFGZ ::= "if>0"
IFLEZ ::= "if<=0"
IFLZ ::= "if<0"
IFNEZ ::= "if~0"
IFEZ ::= "if=0"
IFT ::= "ift"
IFF ::= "iff"
IFE ::= "ife"
IFNE ::= "ifne"
FUNCTION ::= "fun"
BIND ::= "bind"
ALSO ::= "also"
THEN ::= "then"
EXPORT ::= "export"
IMPORT ::= "import"
LAMBDA ::= ",\"
```

1.2 Punctuations

Punctuations used in the Towel programming language are as follows:

- Whitespace characters are simply ignored.
- These characters have special meanings in the Towel programming language: `` ` ' `` , ; ( ) [ ] \ @ EOF. This means that you cannot use these characters in names and atoms. [1]
- Any unprintable character is reserved and won't be used.

Lexeme Definition

```
unprintables ::= [all the unprintable ASCII characters]
```

[1] In other words, you can use any other punctuation characters in names and atoms.

```
whitespaces ::= ['\n' '\t' ' ' '\r']
reserved_punct ::= ['`' ''' '"' ',' '\' '@' '.'
                    '(' ')' '[' ']' '{' '}' whitespaces unprintables]
valid_punct ::= ['!' '~' '#' '$' '%' '^' '&' '*' '-' '_' '+' '=' '|' '.'
                 ':' '<' '>' '?' '/' ';']
BQUOTE ::= '`'
SQUOTE ::= '''
DQUOTE ::= '"'
COMMA ::= ','
SLASH ::= '\'
AT ::= '@'
LPAREN ::= '('
RPAREN ::= ')'
LBRACKET ::= '['
RBRACKET ::= ']'
LBRACE ::= '{'
RBRACE ::= '}'
```

1.3 Names

Names are used for naming (or to be more precise, referencing to) values. Valid names should not start with reserved punctuations, lowercased letters, and numbers.

More formally,

Lexeme Definition

```
digit ::= ['0'-'9']
hexdigit ::= ['0'-'9' 'a'-'f' 'A'-'F']
bindigit ::= ['0'-'1']
lc_chars ::= ['a'-'z']
NAME ::= [^ '-' reserved_punct digit lc_chars] [^ reserved_punct]*
```

1.4 Literals

Most easy-to-use languages support a wide variety of literals (Python is a good example and Java is not). The Towel programming language supports literals for atoms, integers (fixed, unsigned fixed, big), floats, strings, lists and tuples. They are defined as follows (rule for list literals will be revealed later):

Lexeme Definition

```
ATOM ::= lc_chars [^ reserved_punct]*

signed ::= ['+' '-']
fint_body ::= (("0d"? digit+) | ("0x" hexdigit+) | ("0b" bindigit+))
FINT ::= signed? fint_body
INT ::= signed? digit+ ['L' 'l']
UFINT ::= '+'? fint_body ['U' 'u']

dot ::= '.'
int ::= digit+
frac ::= digit+
exp = 'e' signed? int
dot_float = ((dot frac) | (int dot frac)) exp?
```

```
exp_float = int (dot frac)? exp
FLOAT ::= signed? (dot_float | exp_float)

string_char ::= [^ '\' ''']
string_esc_seq ::= '\' string_char
string_item ::= string_char | string_esc_seq
STRING ::= ''' string_item* '''
(Rules for strings is from the lexical parsing section of the Python
 language reference manual.)
```

Implementation Detail

Because positive (or negative) numbers, "+1." for example, also uses the plus symbol, the rule for `NAME`s is actually more complicated than what's written above:

```
NAME ::= valid_upper_char common_valid_char*
       | '+' common_valid_char_no_digits?
       | '+' common_valid_char_no_digits common_valid_char*
       | '-' common_valid_char_no_digits?
       | '-' common_valid_char_no_digits common_valid_char*
```

1.5 Comments

Comments are defined as follows:

```
__COMMENTS ::= '"' [^ '"']* '"'
```

1.6 Lexical Error

When the scanner encounters any other character not mentioned above, it will raise a `LexicalError` exception.

Chapter 2

Data Types

This chapter covers the basics on types in Towel. It worth mentioning here that all the values of whatever types of Towel is immutable.

2.1 Built-in Types

Towel provides to the user the following primitive built-in types:

- Atom
- Fixed integer or, FixedInt
- Unsigned fixed integer or, UFixedInt
- Big integer or, Int
- String
- Float
- List
- Tuple

2.1.1 Functions

Functions are one of the most important kind of values in Towel. They can be returned as values, passed as arguments, and evaluated as regular functions. This kind of feature is often called as *functions as first-class citizens*. A function gets its argument from caller's data stack (often the data stack beneath it, or the same stack if it's a tail recursive call), and returns the TOS on its data stack.

Hint

Although it is not recommended, you can always use the `!!pack\Std` routine[a] to pack multiple values in the callee and `!!unpack\Std` them in the caller to achieve multiple return values.

[a] A routine in Towel is a function with side-effects (or stack-effects).

2.1.2 Atoms

Atoms are special names uniquely bound to integer constants. But they are not comparable to integers, nor can they be applied to numeric operations. It's also meaningless to compare between two atoms.

Towel predefines `false` and `true` as boolean atoms.

Implementation Detail

Although they are not comparable to integers, they can be tested against each other to see if they are the same. The earlier the atom appears in code, the smaller the integer constants it gets.

2.1.3 `(U)FixedInts`, `Ints`, `Floats`

(Unsigned) fixed integers and floats are 64bit integers and floats. `Int`s are signed integers of arbitrary precision (like those `int`s in Python). These types are said to be subclass of the class `Number` (only conceptually), which is to say all these types are supported by basic arithmetic operations. However, bitwise operations will only take (`U`)`FixedInt`s as arguments.

2.1.4 Strings

A string is an enumerable data structure of a sequence of characters (or bytes). With that said, you can operate string with most of the list operations, for example, `#hd\Std` and `#cons\Std`.

Implementation Detail

To see how long a string can be in the underlying OCaml environment, go to your OCaml toplevel and type `Sys.max_string_length;;`.

2.1.5 Lists and Tuples

Lists and tuples are enumerable types in Towel.

Lists are accepted by list-related functions, such as `#hd\Std`, `#tl\Std`, `?#\Std` (list emptiness test). However, note that only lists are supported by the `#cons\Std` operation. When you `#cons\Std` to a tuple, Towel Virtual Machine will blow up.

Tuples are accepted by `#tn\Std` to access specific element of them. They can also be tested emptiness with `?#\Std`.

2.2 Conversion between Types

Most of the operations can only deal with homogeneous data types. For example, addition can only happen between two fixed integers, or two unsigned fixed integers, or big integers, etc. A addition between a float and an integer will result in error.

If you really would like to add a float to an integer, either use the built-in function `~float\Std` to convert the integer into a float, or use `~int\Std` to convert the float into an integer.

2.3 Runtime Type Reflection

You can use the built-in function `^?\Std` to get the type of TOS. The returned type is a built-in value that represents types. Type values can be test against each other for equality with `=\Std`.

Chapter 3

Hello, Word

3.1 Program Structure

Grammar

```
sentence : word* TERMINATOR

word : backquote
     | sequence
     | literal
     | control_sequence
     | function_
     | bind_sform
     | import
     | export
     | name
```

A Towel program is a sentence that consists of one or multiple so-called **words** or **forms**.

When encountered multiple words, they are always evaluated one by one in the order they appear. Although most of the times, Towel remains in a postfix fashion, but for the sake of convenience, some parts of the grammar is of prefix or infix style (e.g. the bind form and namespaced name invocation).

A word can be one of the following:

- literal
- name
- sequence
- backquote
- `if` forms
- function form
- bind form
- import and export form

You should also know that the computational model Towel uses is based on stacks, or to put it more precisely, stack of stacks. That is to say, when you invoke a function, a new stack in created for that particular function, and after it returns the new stack gets destroyed. This avoids potential corruption of only a single stack.

3.2 Rules for Evaluation

When you reference to words, Towel does the following for each kind of them:

- **Literals**

  pushes back them directly

- **Backqoutes**

  pushes back whatever is quoted (i.e. without evaluating)

- **Functions**

  creates a new stack, gets whatever required arguments to it from caller's stack, does the computation, pushes back the TOS of its own stack to caller's stack (this is called in Towel "returning a value")

- **Sequences**

  creates a new function out of the body of the sequence and evaluates that function

- **Names**

  Grammar

  ```
  name : NAME (SLASH NAME)*
       | name AT ;; tail recursive call
  ```

  looks up the value it references to, evaluates that value and pushes back the evaluated value

- **`if` forms**

  tests against the TOS and evaluates the word in respective branch

- **`bind-then` forms**

  pushes a new scope, evaluates the values on stack that get bound (both `bind` clause and `also` clauses) and binds the values evaluated to the names, then evaluates the `then` clause within current scope, and finally pops the scope

3.3 Rules for Scoping

Only two forms in Towel can create a new scope: the function form and the `bind-then` form. This means that a scope is created when entering a function, or a `bind-then` form, and implies that a scope is destroyed when exiting a function, or a `bind-then` form.

A name is only referenceable within the scope it is bound, plus the child scopes of that scope. For closures, the captured names are poured into the function's scope by default.

3.4 Literals

Grammar

```
literal : LITERAL
        | STRING
        | ATOM
        | lit_list
        | lit_tuple
```

A literal is a literal value whose type is of the data types we have talked about in chapter 2.

3.4.1 Atoms

You can create atoms by writing any lowercased letter followed by arbitrary length of characters that are not reserved punctuations and keywords.

Atoms are unique across the entire program. Because atoms are assigned with a unique unsigned 64-bit integer, so you can have no more than 2^{64} of them in your program.

Implementation Detail

Boolean type is implemented as atoms. `:and\Std`, `:or\Std`, `:not\Std` work for integers as well as booleans.

3.4.2 Numbers and Strings

You can create number and string literals by writing like this:

Example

```
1 -1 2 -3 5 -8 13 -21
+42424242424242424242424242424242424242424242424242424242L
-3141592653589793238462643383279l
3U +0x4579u 0b11101 0d1234
1.1e1 -0.1 'don\'t panic'
```

Note that string items are surrounded by single quote, rather than double quote.[1]

3.4.3 Lists

Grammar

```
lit_list : LBRACKET word* RBRACKET
```

When creating a list literal, you must write a list of words separated with spaces in a pair of brackets, like the following code:

Example

```
[arthur-dent ford-prefect betelgeuse]
[Spam Spam Spam]
[Spam ifne (More Spam)`, (Less Spam)`] (valid but bad example)
```

Be warned!!

Although you can safely write list literals with names, and literals (integers, atoms, or even list), but because of Towel's stack-based nature, complicated words in a list literal may not turn out to be what you want.

When the Towel compiler compiles a list literal into instructions, it first generates a pair of instructions called `push-lnil`, `end-list`. Then it compiles the words being the content of the list literal, and put the generated code between that pair of instructions. But be warned that, when entering the `push-lnil` and `end-list` pair, the target of most of the pushing actions will be come the newly pushed list, rather than the data stack. For example,

```
push-lnil
push-lit 1u
push-lit 2a
```

[1] Because you don't have to hit the *shift* key when inputing single quotes. Same goes for brackets.

```
push-lit 3l
end-list
```

will result in `[1 2 3]`. And

```
push-lnil
push-lit 1u
push-lnil
push-lit 2a
push-lnil
push-lit 3l
end-list
end-list
end-list
```

will result in `[1 [2 [3]]]`. Just like what we expected, right?
But consider the following,

```
(2 3 [1 2 +] !println)
```

This is syntactically correct Towel, but the problem is that it tries to evaluate a function within the environment of creating a list. The user might expect the output being `[3]` but that not the case: before the name `+`, the stack is like the following,

```
{| 2 3 [1 2 |}
```

when the addition function tries to grab its arguments, it first pop the unfinished `[1 2` as its argument, which is incorrect already. And it then gets its second argument from the stack, which is obviously 3.
You should also be very cautious with

```
(1 2 [3 if>0 +`, -`])
```

This `if>0` is tested against the TOS, which is the unfinished list `[3`!
So in general, never ever put complicated words in a list literal. If you are really forced to do so, enclose the word in a sequence (parentheses).

3.4.4 Tuples

Grammar

```
LBRACKET SLASH word* RBRACKET
```

Tuples are fixed length lists, this means that you cannot CDR from them, nor CONSing to them. Create tuples like this:

Example

```
[\ arthur-dent ford-prefect betelgeuse]
[\] "an empty tuple"
```

Just so you know, the warning of the previous section on lists applies here as well.

3.5 Sequence

Grammar

```
sequence : sequence_
         | shared_sequence

sequence_ : LPAREN word* RPAREN

shared_sequence : LPAREN AT word* RPAREN
```

Sequences are short-hand forms for creating anonymous functions with no arguments. You can create a sequence by writing the sequence body between a pair of parentheses.

Towel also provides another kind of sequences, the shared sequences. This kind of sequences share the same context (such as stack and scope) with the caller. When creating such sequences, you add an at symbol right after the left parenthesis.

Example

```
((A B - if>0 1, 0) (A B + if<0 2, 3) :and) "non-shared regular sequences"
(A B - (@ if>0, 1, 0) !println) "prints 1 or 0"
```

Hint

Just for your information, the Towel compiler automatically strips off the creation of certain unnecessary sequences for performance reasons. For example, the following sequences will be optimized out.

```
bind THE-ANSWER 42
then (THE-ANSWER !println)

42 if>0 ('The answer is greater than void.' !println),
  ('The answer is less than void.' !println)
```

The appearances of the sequences above are simply only of syntactical reasons, namely, to avoid ambiguity of the grammar. They can be safely removed semantically.

See section 3.12 for more details.

3.6 Backquote

Grammar

```
backquote : literal BQUOTE
          | name BQUOTE
          | sequence BQUOTE
          | backquote BQUOTE
          | LBRACE word* RBRACE ;; shorthand for backquoted shared sequence
```

Towel evaluates and pushes everything it encounters, you can use backquotes the values to prevent Towel from evaluating them so that Towel pushes them directly onto the data stack. Backquotes are created by appending a backquote to the words you want to backquote.

You can backquote only limit types of words:

- **Literal**

  Towel pushes the literal back immediately

- **Name**

  Towel pushes whatever the name references to onto the stack, without evaluating them

- **Sequence and function**

  Towel pushes them onto the stack without executing it

- **Backquote**

  Why would you do such a thing?

Hint

You can create backquoted shared sequence by replacing the parentheses with braces and dropping both the at symbol and backquote. See also subsection 6.2.2 and subsection 6.2.3.

3.7 `if` Forms

Grammar

```
control_sequence : if_sform

if_sform : IFGEZ word COMMA word
         | IFGZ word COMMA word
         | IFLEZ word COMMA word
         | IFLZ word COMMA word
         | IFE word COMMA word
         | IFNE word COMMA word
         | IFEZ word COMMA word
         | IFNEZ word COMMA word
         | IFT word COMMA word
         | IFF word COMMA word
```

Towel supports 10 kinds of `if` forms for the sake of readability and convenience. They are of the same form, while differing in the predicate they use.

An `if` form contains two words separated by a comma. When evaluating an `if` form, Towel tests the TOS and see if it satisfies the condition. If the condition is satisfied, the first word (called the true branch) is evaluated and the second word is ignored[2], and vice versa. By default, `if` forms does not consume TOS, see section 3.12 for more detail.

The predicates used by `if` forms are as follows:

- `if>0`

  if TOS is a number and greater than 0

- `if>=0`

  if TOS is a number and greater than or equal to 0

- `if<0`

  if TOS is a number and less than 0

- `if<=0`

  if TOS is a number and less than or equal to 0

- `if=0`

  if TOS is a number and equal to 0

- `if~`

  if TOS is a number and not equal to 0

[2]This is basically why you want a designated condition form

- `ife`

  if the stack is empty

- `ifne`

  if the stack is not empty

- `ift`

  if TOS is an atom and equal to `true`

- `iff`

  if TOS is an atom and equal to `false`

See chapter 6 for examples on `if` forms.

3.8 Function Form

Grammar

```
function_ : FUNCTION arg_def* COMMA word
          | FUNCTION BQUOTE arg_def* COMMA word
          | LAMBDA arg_def* COMMA word ;; same as FUNCTION BQUOTE

arg_def : NAME
```

Function forms are used to define and *immediately* execute anonymous functions of arbitrary arity. To do this, first type the keyword `fun`, and a list of argument declarations and finally a word for the body of the function. The function will acquire the declared arguments from the stack in the reverse order as the argument declarations indicate.

Because function form creates and evaluates function in place, the following code is valid[3]:

Example

```
fun A B,
  (A B fun X, (3 X +))
```

In practice, you may want to use the `bind-then` form and backquote jointly to create functions. You use backquote to prevent the function from evaluating so that you can use it later, for example, in the `then` clause.

Example

```
bind Some-practical-function fun` ~arg1 ~arg2 ~arg3,
  (~~do-something-with-the-arguments~~)
then (Some-arg1 Some-arg2 Some-arg3 Some-practical-function)
```

Hint

You can also use the punctuation sequence "`,\`" to replace `fun`.

See chapter 6 for concrete examples.

[3]But not semantically correct, because you cannot invoke a number as function.

3.8.1 Tail Recursive Function Calls

Any practical functional programming language provides tail recursion optimization. So does Towel. However, Towel is unable to identify[4] whether a function call is tail recursive, so users are responsible for tagging tail recursive calls with an **at** symbol at the end of the name of the function, like this:

Example

```
bind Loop fun` F It End,
  (It End - if=0 (It F),
              (It F F` It 1 + End Loop@)
then (("looping" !println)` 1 10 Loop).
```

3.8.2 Partial Function Application

When applied with insufficient number of arguments, Towel will return a partially applied function instead of doing all the computation it is supposed to. To be more precisely, Towel copies the function being called as a new value, install the arguments already applied to that new function value, then return this new function value as the return value of the old function.

Example

```
bind ~dec-1 (1 -)
then ('The Answer is ' !print 43 ~dec-1 !println)
```

3.8.3 Phony

A phony in Towel, is a special value on stack. When a phony is the current TOS, the stack will be considered empty. To push a phony onto the current stack, use `$$\Std`.

Example

```
($$ ife (true !println), (false !println))
```
This will print `true`.

Often, phony is used in conjunction with partial function application to create a partial function.

Hint

Note that every phony will be popped by the TVM right after it is used, otherwise it will shadow anything pushed before it. For example the following code can produce expected answer because of this feature:

```
([1 2 3 4 5] $$ 3 < /filter !println)
```

The output will be `[4 5]`.

3.9 Bind Form

Grammar

```
bind_sform : BIND bind_body (ALSO bind_body)* THEN word

bind_body : NAME word
```

[4] *I admit I had been lazy.*

Use bind forms to add new name bindings in a new scope. Names can be bound to any kind of values such as functions, atoms, and all kinds of literals as long as they exist on the stack or is exported by the respective module.

Simply type `bind` followed by the name and the value. Use keyword `also` to bind more names to more values. Bind forms have a compulsory `then` clause, which is followed by a word. You can do your computation under the name scope after this name binding in the `then` clause. Top-level name bindings, i.e. names bound by the outmost bind form, are visible across modules, you may want to take advantage of this behavior.

Example

```
bind A 40
also B 2
then (A B + !println)
```

Implementation Detail

Binding in TVM is currently implemented as names associating to the absolute index of the values in the stack. See also the following warning.

Be warned!!

Note that all the computations are done on stacks, so are bindings. That is to say that, the values that get bound exists on stacks. So you may want to be careful about this.

Fasten your seat belts for the next two examples.

```
bind Answer 42
then (!!pop Answer !println)
```

will result in your program blowing up, because the position where there used to hold the value for the Answer is now invalid.

What's more scare is this,

```
bind Answer 42
then (!!pop 41 Answer !println)
```

You just accidentally changed the value of the Answer without even noticing it!

3.10 Module

Modules, or namespaces, in Towel, are sets of names. Mechanisms like this prevent names of various files from colliding into each other.

3.10.1 Importing a Module

Grammar

```
import : IMPORT STRING* SLASH ;; explicit import
       | IMPORT STRING* AT ;; implicit import
```

To import a module, use the `import` form.

Example

```
import 'std' \
import 'std' @
```

The first `import` form means importing the module defined in a file named `'std.w'`, and name that module `Std` so that you can reference to the names from this module as `SomeName\Std`. The ending punctuation backslash is exactly the samething you use for module referencing. This is called explicit importing.

The second `import` form ending with an at symbol, means not only importing the module `Std`, but also importing all the exported names in that module. In this way, you are allowed to reference to the names without having to specify the modules they are in. This is call implicit importing.

Implementation Detail

During compilation, the file the compiler searches is a `'.e'` file, this file contains only all the names available in the respective module.

During execution, however, the file the virtual machine needs is a `'.w'` file which is the compiled version of the actual program.

Also note that in current version of TVM, implicit importing imports and pushes all the exported values from the imported module onto the stack. So you may expect a non-empty stack after implicit importing.

Implementation Detail

Be aware that imports affects exactly the current scope, and the children scopes of the current one. When current scope exits, the side-effects the import form brought about are also gone.

This also implies that values pushed by implicit importing are popped along with the stack being destroyed.

Hint

For your imformation, no module is imported by default. You have to explicitly import any modules you want to use.

The path of the module file `'std.w'`, is searched as the following order:

- the current working directory
- the predefined directory `towelibs`

Implementation Detail

In current implementation, you can modify both the `'compiler/config.ml'` and `'vm/config.ml'` and recompile them to change the default search paths. We may add a environment variable for search paths in the future.

Note that you can import multiple modules in a single `import` statement.

3.10.2 Exporting Names

Grammar

```
export : EXPORT NAME* AT
```

You can also export names in your module so that they are visible to other modules with `export` form as follows:

Example

```
bind A 1
also B 2
also C 3
then export A B C @
```

The Towel compiler will automatically generate a `'.e'` file for each compiled `'.t'` file for the purposes of name exportation.[5] An empty one will be generated, even if there are no names exported in the source file.

3.11 Wait, What about Garbage Collection?

Towel definitely collects garbage for you!

In current implementation of the TVM, when a value is popped out from a stack, the value gets GC'ed immediately.[6] When a stack is popped from the data stack stack, all the values on the popped stack are GC'ed immediately. So basically, Towel gets garbage collection for free because of its stack-based computation model.

Oh, and of course, extensions have to manage their own memory. Because they are out of the TVM's touch.

3.12 Switches

Towel provides some switches to change the default behavior of the compiler:

- `hungry`
- `share-stack`
- `optimize-seq`, on by default

If you want to turn on/off these switches, type in the switch names on first line, leave an empty line next to it, then go on with your code, like this:

Example

```
I'm so hungry! Also please share-stack.

bind Something-new (1 2 -)
then Something-new
```

If `hungry` is turned on, `if` forms will be compiled to their respective hungry versions, which consume the TOS they test against when the test finishes, i.e. immediately after falling in the true branches. This is useful when you want to be thrifty about stack spaces.

Hint

You can also achieve this with `hungry` turned off by calling `!!pop\Std` explicitly.

When `share-stack` is on, every function (including sequence of course) uses the same stack, you get more classic stack-based language programming experience[7] out of this switch, but you may want to make sure functions don't leave extra elements on the stack, so you'd better turn on `hungry` switch along with this.

[5] Very much like the C header files, but less powerful.

[6] Mayn thanks to OCaml!

[7] And probably faster execution, because the context switching is done a lot faster without data stacks pushing and popping.

If you turn on `optimize-seq` switch, when you create a sequence as the body of a function, `if` form, or `then` clause of `bind-then` forms, this sequence is optimized to disappear, leaving the body of it as the body of the form.

Implementation Detail

The reason that I don't make them arguments to the compiler, is because I believe these kinds of configuration is a part of the code. If you were to make the arguments to the compiler, users with different compiler settings while compiling will have different bytecode, thus different result.

Chapter 4

The Standard Library

4.1 Naming Conventions

In towel, we encourage Lisp-like naming styles. But since only upper-cased characters are allowed to appear as the first alphabet character in a name, we normally prepend a punctuation to it.

- For routines, we would normally prepend an exclamation mark to such routines. For example, `!print\Std`. For routines with serious side-effects, we prepend two.
- For functions that operate on enumerables, we prepend `#`.
- For functional functions, we prepend a slash (half of the λ character). For example, we have in the standard library `/foldl\Std`, `/map\Std`.
- For predicates (something that returns `true` or `false`), we prepend a question mark.
- For arithmetic functions, we don't prepend anything.
- For arguments, we usually prepend a `~` for the sake of simplicity.

4.2 Module `Std`

Module `Std` is the very basic standard library of the Towel programming language. It consists of functions of multiple domains.

4.2.1 Arithmetic Functions

This category contains the following functions:

- `~1 ← ~2 ← +\Std`
  add `~1` and `~2`
- `~1 ← ~2 ← -\Std`
  `~1` minus `~2`
- `~1 ← ~2 ← *\Std`
  multiply `~1` and `~2`
- `~1 ← ~2 ← /\Std`
  divide `~1` by `~2`
- `~1 ← ~2 ← %\Std`
  `~1` modulo `~2`

Hint

This leftarrow thing is just to highlight the arguments that should be applied to the function, and implies the stack-based nature of Towel, i.e. the order the functions acquire their arguments.

For example, `~1 ← ~2 ← -\Std` means that although `-\Std` grabs `~2` first, then `~1`, it still calculates `~1 − ~2` (and is defined as `fun` ~1 ~2` if you look at the source code of `std.t`).

It also contains various functions for comparison purposes:

- `~1 ← ~2 ← =\Std`
  tests if `~1` equals to `~2`
- `~1 ← ~2 ← >\Std`
  tests if `~1` is the greater than `~2`
- `~1 ← ~2 ← >=\Std`
  tests if `~1` is the greater than `~2`
- `~1 ← ~2 ← <\Std`
  tests if `~1` is the less than `~2`
- `~1 ← ~2 ← <=\Std`
  tests if `~1` is the less than or equal to `~2`
- `~1 ← ~2 ← <>\Std`
  tests if `~1` is not equal to `~2`

Let's not forget about bitwise arithmetic functions:

- `~1 ← ~2 ← :and\Std`
  bitwise and between `~1` and `~2`; (works for `true` and `false` too)
- `~1 ← ~2 ← :or\Std`
  bitwise or between `~1` and `~2`; (works for `true` and `false` too)
- `~1 ← ~2 ← :not\Std`
  bitwise not operation on `~1`; (works for `true` and `false` too)
- `~1 ← ~2 ← :xor\Std`
  bitwise xor between `~1` and `~2`
- `~x ← ~n ← :shl\Std`
  bitwise shift left `~x` for `~n` bits, `~n` is an unsigned fixed integer
- `~x ← ~n ← :shr\Std`
  bitwise shift right `~x` for `~n` bits, `~n` is an unsigned fixed integer
- `~x ← ~n ← :lshr\Std`
  bitwise logical shift right `~x` for `~n` bits, `~n` is an unsigned fixed integer

Hint

Arithmetic functions only work for numbers.

The equality and non-equality function works for most of the types (including type `Type`). The comparison functions only work for numbers because they are implemented using the substraction

function.
Bitwise functions only work for integer types.

Hint

The above functions (except bitwise shifting ones) must be applied to values of the same type. Otherwise the TVM will throw an error and exit.

4.2.2 Conversion Functions

Because Towel is a strong-typed language, we provide you some conversion functions to cast the values around.

Be warned!!

Think before you use these functions.

- `~1 ← ~fint\Std`

  converts `~1` of unsigned fixed integers, floats, big integers, or strings into a fixed integer

- `~1 ← ~ufint\Std`

  converts `~1` of unsigned fixed integers, floats, big integers, or strings into an unsigned fixed integer

- `~1 ← ~int\Std`

  converts `~1` of unsigned fixed integers, floats, big integers, or strings into a big integer

- `~1 ← ~float\Std`

  converts `~1` of unsigned fixed integers, floats, big integers, or strings into float number

- `~1 ← ~str\Std`

  stringifies `~1` of any possible value to a string

4.2.3 Reflection Function

Use `^?` if you want to know the type of a value at runtime. The return value of this function can be tested equality against other type values.

4.2.4 Routines

- `~x ← !print\Std`

  prints `~x`

- `~x ← !println\Std`

  prints `~x` plus a newline character

- `!read\Std`

  reads in a string from standard input and push it onto the stack

- `!!pop\Std`

  pop the last element on current stack

- `!!dup\Std`

  duplicate the last element on current stack

- `... ← ~n ← !!pack\Std`

  pack `~n` elements on current stack into a list, if `~n` is negative one, it packs all the elements on the stack until it encounters a phony (in this case, the phony is popped as always)

- `~1 ← !!unpack\Std`

  unpack `~1` into N elements and push them onto current stack

  Be warned!!

  You may want to be cautious when `!!pack\Std`'ing −1 on a stack that recently ran implicit import. You could pack every function that's imported and placed on the stack into a list, rendering the name bindings invalid.

  Always using a phony when applying −1 to `!!pack\Std` is a good idea.

- `~fun ← !invoke\Std`

  invoke `~fun` as a function when you don't know the arity of it

Hint

- Because of the low-levelness of `!!pack\Std` and `!!unpack\Std`, they do not accept argument the same way other functions do, but directly from current stack. So you cannot create partially applied functions out of them.
- You can use `!!pack\Std` to implement variadic functions.

4.2.5 Phony

Use `$$\Std` to push a phony onto current stack.

Be warned!!

Do not try to `!!pop\Std` a phony, it will result in an `PhonyEmptyStack` error.

4.2.6 Working with Enumerables

Functions that Works with Lists, Tuples and Strings

Only one exists for now: `~x ← ?#\Std`, `~x` emptiness test.

List and String only Functions

- `~l ← #hd\Std`

  get the head of `~l`

- `~l ← #tl\Std`

  get the tail of `~l`

- `~e ← ~l ← #cons\Std`

  cons `~e` and `~l` into a new list or string if `~l` is a string

- `~l ← #rev\Std`

  reverse `~l`

- `~l1 ← ~l2 ← #concat\Std`

  concatenate `~l1` and `~l2`, for strings it's the same as `#cons\Std`

- `~l ← #len\Std`
  get the length of `~l`

Hint

Remember everything in Towel is immutable.

Tuple-only Functions

- `~n ← ~t ← #n\Std`
  get the `~n`-th element of `~t`
- `~t ← #1\Std`
  get the first element of `~t`
- `~t ← #2\Std`
  get the second element of `~t`
- `~t ← #3\Std`
  get the third element of `~t`

4.2.7 Functional Functions

- `~x ← /id\Std`
  the identity function, i.e. it returns `~x`
- `~init ← ~list ← ~fun ← /foldl\Std`
  folds the list`~list` from left to right on function `~fun`, with the initial value being `~init`
- `~init ← ~list ← ~fun ← /foldr\Std`
  same as `/foldl\Std`, except that this walks the list from right to left and is not tail recursive function
- `~list ← ~fun ← /map\Std`
  maps function denoted by `~fun` onto `~list`
- `~list ← ?pred ← /filter\Std`
  for each element `~x` in `~list`, discards `~x` when `~x ?pred` turns out to be `true`
- `~args ← ~fun ← /apply\Std`
  apply the elements in `~args` as arguments to `~fun`

Hint

You may want to use backquotes when using these functions, for example,

```
([1 2 3 4 5] !println /map)
```

prints out

```
[1 2 3 4 5]
```

rather than

```
1
2
3
```

```
4
5
```

To do this correctly, you should backquote `!println` like this

```
([1 2 3 4 5] !println` /map)
```

This backquote prevents `!println` from executing, leaving it be as a function value so that it can be passed to `/map`.

4.2.8 Variadic Functions

Most of the variadic functions are implemented using one of the features of `!!pack\Std` function, i.e. it packs all the elements before a phony into a list if `~n` is -1.

- `~f ← ~n ← /arg-rot-n\Std`

  rotates the last `~n` arguments of `~f`

  Example

  ```
  import 'std' @

  bind Some-fun ,\ ~1 ~2 ~3, (~1 ~2 ~3 + *)
  also Rot-some-fun-1 (Some-fun` 3 /arg-rot-n)
  also Rot-some-fun-2 (Some-fun` 2 /arg-rot-n)
  then (1 2 3 Some-fun !println
        1 2 3 Rot-some-fun-1 !println
        1 2 3 Rot-some-fun-2 !println)
  ```

  The above code will print out

  ```
  5
  9
  5
  ```

  We get the second output 9 is because after `3 /arg-rot-n`, the function `~1 ← ~2 ← ~3 ← Some-fun` becomes `~3 ← ~1 ← ~2 ← Some-fun`, so the result is `3 1 2 + *`, which is 9.

  THe third output is still 5 because the operands of addition is interchangeable, despite the fact that `Some-fun` becomes `~1 ← ~3 ← ~2 ← Some-fun`.

- `~f ← /arg-rot3\Std`

  a special version of `/arg-rot-n\Std` that works with trinary functions

- `~f ← /flip\Std`

  a special version of `/arg-rot-n\Std` that works with binary functions

- `... ← #!vargs\Std`

  accumulates arguments into a list until it encounters a phony, effectively -1 applied to `!!pack\Std`

- `... ← ~acc ← ~f ← /!vfoldl\Std`

  a variadic version of `/foldl\Std`

- `... ← #!vconcat\Std`

  concatenates all values (assuming they are of all concatenable data types) on the stack until it encounters a phony

4.3 Module `Random`

This module exploits the extension mechanism provided by current implementation of the TVM to easily invoke functions in OCaml. See the virtual machine manual for more details.

The provided functions are

- `~s` ← `~seed\Random`

  change the seed used in random number generation to `~s`

- `~useed\Random`

  change the seed according to `/dev/urandom` on *nix systems, or system parameters if `/dev/urandom` is not available

- `~~\Random`

  generates a random floating point number within the range of $[0.0, 1.0]$.

Chapter 5

Running a Towel Program

5.1 Weaving a Piece of Code

The compiler for the Towel programming language is codenamed `weave`, you can find it in `build/src/compiler` after successfully compiling the compiler portion of the project with

Example

```
./waf configure build --compiler --native
```
**This will build a standalone compiler executable.*

To compile a source file `foo.t`, and want the compiled output to be `bar.w`, use

```
path-to-weave foo.t -o bar.w
```

5.1.1 Advanced Usages

If you want to see the human-readable bytecode file, apply -t to `weave`, for example

```
path-to-weave foo.t -o bar.l -t
```

If you want to read the unassembled compiled file (human-readable) for debugging purposes, apply -r to `weave` like this

```
path-to-weave foo.t -o bar.o -r
```

If you have a human-readable bytecode file, and want to compile into VM-readable bytecode file, apply -b to `weave`, e.g.

```
path-to-weave foo.l -o bar.w -b
```

Hint

Recommended extension for Towel source file is `.t`, because it's the first letter of the word *towel*.

Recommended extension for raw unassembled compiled file is `.o`, which means original Towel.

Recommended extension for Towel bytecode file is `.w`, meaning woven Towel. It's also the third letter of *towel*![a].

By default, files that contains the exported names of the module have the extension of `.e`, meaning exportation.

And finally recommended extension for human-readable bytecode file is `.l`.

[a] I love wordplay so much!

5.2 Executing the Woven Binary

You will have to build the Towel Virtual Machine before executing a bytecode. You can locate it in `build/src/vm` after compiling the VM portion of the project with the following commandline:

Example

```
./waf configure build --tvm --native
```
**This will build the standalone Towel Virtual Machine executable.*

Hint

Normally you may want to apply `--native` to Waf to build a native version of the Towel Virtual Machine, otherwise the average loading time and execution time will be longer.

Use `./waf test` with native builds and non-native builds to feel the difference.

With the `tvm` executable available, you can run your bytecode-compiled programs with

```
path-to-tvm foo.w
```

To debug your program, use `-t` to trace the execution of the virtual machine.

```
path-to-tvm foo.w -t
```

Hint

The output size may be enormous for even a small piece of code. We recommend you to redirect stderr to a file, and analyze the problems with that file.

```
path-to-tvm foo.w -t 2> trace.log
```

Chapter 6

Examples

The following examples run under the default switch configuration.

6.1 Concrete Examples

6.1.1 Greatest Common Divisor

Example

```
import 'std' @

bind GCD fun` A B,
  (A B - if=0 A (5),
     if>0 (1) (!!pop A B - B GCD@ (2)),
     if<0 (!!pop (3) A B A - GCD@),
       ~idle (4))
then (42 24 GCD !println)
```

1. See how `if` forms are chained here!
2. Note that we are tail recursing the `GCD` function here.
3. As is mentioned in previous chapters, when doing tail recursive calls, you really want to be careful about your stack usage to avoid stack leakage.
4. Remember an `if` form have two branches, and both of them cannot be omitted. So we put ~~Eric Idle~~ `~idle\Std` here to denote that we don't want to do anything here, just a placeholder.
5. The exit of the recursive function, so we'll directly put `A` here, without popping the result of `A B -`. Because this stack will be destroyed and all the values on it GC'ed, after it returns `A`.

6.1.2 Fibonacci Numbers

Example

```
bind Fib fun` A B N,
  (N if=0 A,
       (A B + A 1 N - Fib@))
then (1 1 10 Fib)
```

Trivial.

6.2 And Now for Something Completely Different...

6.2.1 Quicksort

Example

```
import 'std' @ (1)

bind #quicksort (2) ,\ (3) L,
  (L ?#empty ift (!!pop (4) []), (!!pop
    bind ~h (L #hd)
    also ~t (L #tl)
    then (~t (~h >) (5) /filter #quicksort
          [~h]
          ~t (~h <=) /filter #quicksort
      #concat #concat (6))))
then ([5 4 3 2 1] #quicksort !println)
```

1. Implicitly import module `Std` so that we can use the names bound in it without referencing to the module name.
2. Remember our naming conventions? Quicksort works on lists, so we prepend a `#` to it.
3. It's a synonym for `fun`.
4. Use `!!pop\Std` to be thrifty about memory spaces.
5. Utilizing non-shared sequence to create partial applied function.
6. We have three lists on stack, so it takes two `#concat\Std`s to merge them all into one list.

6.2.2 Backquotes

Example

```
import 'std' \

bind Sum ,\ ~ls,
  (0 ~ls +\Std` (1) /foldl\Std)
then export Sum @
```

1. Be sure to quote this plus function (actually, the name) so that `/foldl\Std` can use the addition function, rather than the value the function evaluated, and of course, in this case, you can never evaluate `+\Std` with a number and a list.

6.2.3 (Fake) Macros

Example

```
bind Macro1 (@ if~0 +, -)`
also Macro2 {if~0 +, -}
then bind Fun fun` A B,
       (A B Macro1)
     then (1 2 Fun).
```

A quick explanation: `(@ if~0 +, -)` is bound to name `Macro1` as we want, which is essentially an anonymous function that tests whether the TOS of the caller's stack (because it's a shared sequence) is zero. The overall effect of this piece of code is like we have done a code replacement (at runtime).

`Macro2` is a short-hand version of `Macro1`.

6.2.4 Object-oriented Programming

A Not-even-close One

Example

```
bind Class fun` ~init-value,
  bind ~data ~init-value
  then bind Accessor fun`, ~data (1)
       then [\ (2) Accessor` (3) ] (4)
then bind Instance1-methods (41 Class)
     also Instance2-methods (42 Class)
     then bind Instance1-accessor (Instance1-methods #1 (5))`
          also Instance2-accessor (Instance2-methods #1)`
          then (Instance1-accessor !println
                Instance2-accessor !println)
```

1. `~data` is captured by the `Accessor` function, which will be returned by the `Class` **function** (ha!), and you get all the public methods from the return value of this `Class` function as a tuple.
2. Maybe I should remind you that we use `[\ ]` to denote a tuple
3. Don't forget to backquote you methods!
4. You can replace this `[\Accessor`]` with `(1 !!pack)`.
5. Get the accessor function from the first slot of the tuple.

The above code snippet will produce the output

```
41
42
```

which is exactly what we would be expecting.

A More Advanced Example

Example

```
import 'std' @

bind >>send !invoke`

also Shape ,\ ~type, (7)
  bind __type ~type
  also Type fun`, #1 "dirty hack to make this work with >>send"
  also Area #t2`
  then [\__type [\Type` Area`]]

then bind __meta-Shape ('' Shape)
     also :type (__meta-Shape #t2 #1) (1)
     also :area (__meta-Shape #t2 #2)
```

```
        also Circle ,\ Radius,
          bind __radius Radius (2)
          also ~super ('Circle' Shape) (3)

          then bind Type (~super #1)
               also Area ,\, (__radius !!dup 3.14159 * *) (4)

               then [\Type` Area`] (5)

         also Rectangle ,\ Width Height,
           bind __width Width
           also __height Height

           also ~super ('Rectangle' Shape)

           then bind Type (~super #1)
                also Area ,\, (__width __height *)

                then [\Type` Area`]

         then bind ~my-circle (42 Circle)
              also ~my-rectangle (1 2 Rectangle)
              then (~my-circle :type >>send (6) !println
                    ~my-rectangle :type >>send !println
                    ~my-circle :area >>send !println
                    ~my-rectangle :area >>send !println)
```

1. Make a meta object to get all the available messages a Shape object can receive.
2. A private field of Circle objects.
3. The superclass (or super object).
4. Implementing the abstract method!
5. The public available methods.
6. This is really great: we are sending the message `:type` to the Circle object `~my-circle`, and hoping the object will respond. *Now we are speaking Japanese!*
7. So in retrospect, it's more like an interface or protocol (with some dirty hacks), rather than an abstract class.

The key idea here is that the interface function returns a tuple of the indexing functions that specifiy the position of each method in classes that agree with this protocol.

For example, `:area` is actually the `#t2` function, and is used to get the first element from a Shape object, which is a tuple with the second element being the function that does the calculation for area.

It's a shame that we don't have Lisp macros here, otherwise the syntax will be much simpler here.[1]

6.2.5 GUI Programming

Here in this example, we will show you how to call functions from OCaml using the extension feature of the OCaml Towel Virtual Machine. This part is rather implementation specific. So be sure you are using

[1] Common LISP actually has a package that does object-oriented programming simulation, which is of course much powerful than what I have shown here.

the OCaml implementation of the Towel Virtual Machine.[2] For more on this topic, see also the chapter on Extending the Towel Virtual Machine in the Towel Assembly and Virtual Machine manual.

Below is the code listing of the OCaml-side wrapper for some Tk GUI functions. Notice how this wrapper interacts with the TVM via pushing and popping elements from `dss`, i.e. the data stack stack.

Code Listing

```
open Tk;;
open T;;
open Ext;;
open Nstack;;

let top = ref None;;

let widgets:(int, Widget.toplevel Widget.widget) Hashtbl.t = Hashtbl.create 512;;

let tkfail msg = failwith (Printf.sprintf "TK failure: %s.\n" msg);;

module SimpleTk : TowelExtTemplate =
struct
  let extcall cn dss = match cn with
      1 -> top := Some(openTk ())
    | 2 -> mainLoop ()
    | 3 -> closeTk ()
    | 4 -> update ()
    | 5 -> let s = appname_get ()
      in dspush dss (OVString(s))
    | 6 -> let s = match (dspop dss) with
          OVString(x) -> x
        | _ -> tkfail "unsupported data type for appname_set"
      in appname_set s
    | _ -> tkfail "unimplemented call number"
end

let () = __ext__ := Some(module SimpleTk : TowelExtTemplate);;
```

Then you may want to write a wrapper from Towel-side so that you can call the OCaml wrapper more conveniently.

Code Listing

```
import '.w' \

bind ^~ ('ext_tk.cmo' !>ext\.w)
also !>> !>>\.w`

also >>tk fun`, (1u ^~ !>>)
also ~~~ fun`, (2u ^~ !>>)
also <<tk fun`, (3u ^~ !>>)

also !set-tk-appname fun` ~s, (~s 6u ^~ !>>)

then export >>tk ~~~ <<tk !set-tk-appname @
```

[2] Sure you are now!

The choices of names are purely ideographic. For example, `>>tk` denotes that you are now entering the Tk world, i.e. the `openTk` function of OCaml.

With these two wrappers done, you can write a simple program to invoke the wrappers, like this

Example

```
import 'simple-tk' @

(>>tk
 'Hello, world by the Towel programming language!' !set-tk-appname
 ~~~
 <<tk)
```

You can read this example like this: first we enter the Tk environment, then set the Tk application name to `'Hello, blah blah...'`, after that we call the mainloop denoted by three consecutive tildes. When control returns from the mainloop, we now exit Tk environment by flowing from `tk`. You can of course design the API like this in callback fashion:

Example

```
bind >>with-tk-env ,\ ~fun, (>>tk ~fun ~~~ <<tk)
then (('Hello world!' !set-tk-appname)` >>with-tk-env)
```

If you compile the OCaml-side wrapper correctly and run the last but one example, you will get something like this:

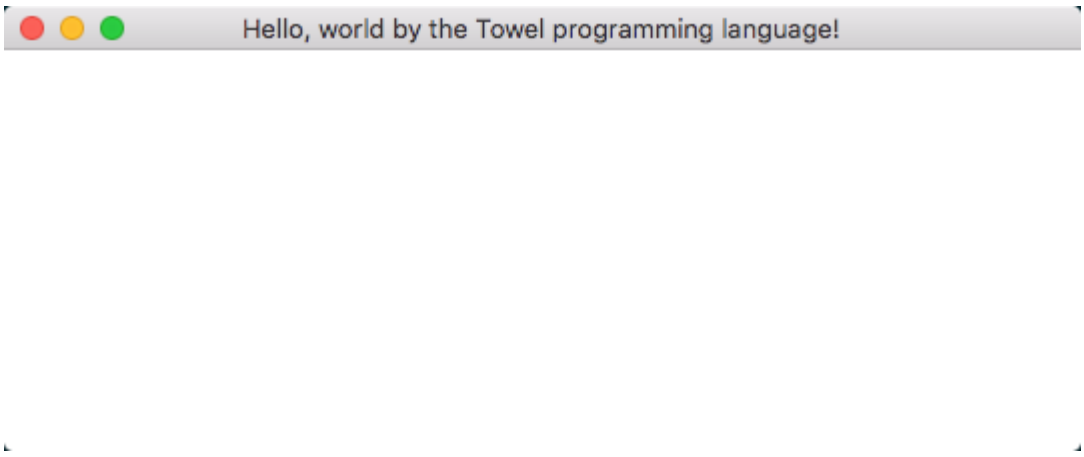

Figure 6.1: Hello world!

Part II

The Towel Assembly and Virtual Machine Manual

Chapter 1

An Overview on the Towel Virtual Machine

The Towel virtual machine is basically a stack-based state machine that accepts and executes Towel Assembly Language. It reads instructions sequently from instruction buffer and does its computation on stacks. It has the following important parts:

- execution stack (or the activation record stack): `ctx_t list`[1]
- data stack stack: `value_t dstack_t dstack_t`
- flags record: `flags_t`
- module table: `(module_id_t, module_t) Hashtbl.t`
- extension table: `(module_id_t, module TowelExtTemplate) Hashtbl.t`

1.1 Execution Stack

Execution stack is where context of function calls are stored. A context of a function call consists the return address of the function call, the module id of the return address is in[2], the function value of the function being called, whether the call is made in a tail recursive fashion, etc.

When a new function call is made, the context of this call is pushed onto the execution stack, followed by the instruction pointer jumping to the start position of the function, and the `curmod` field being switched to the target module. When the instruction `ret` is met, the function returns. TVM pops the TOS on execution stack and jumps to the return address stored in this TOS and switch back to the original module according to the `ctx_t` value.

Implementation Detail

Execution stack has one element in it initially. This context has no meaningful record fields.

1.2 Data Stack Stack

Every function has a stack for itself to do its computation. This avoids stack corruptions along all the function executions. Data stack is essentially an important part of the context of the function call, but it is so important, we would like to operate them manually so that we could be more flexible.

[1]Note that, Towel Virtual Machine does not restricts itself to any single kind of implementation. The use of OCaml type notation here is for convenience and clarity.

[2]The PC jumps around different instruction arrays of different modules.

Normally, when a new function call is made, a new data stack is pushed onto the data stack stack. When the function returns, TVM pops the TOS from the TOS of data stack stack (i.e. the return value of this function is the TOS of current function's data stack), pushes it onto the caller's stack (next to the TOS of data stack stack). Then this data stack gets popped, thus the caller's data stack is now the TOS of the data stack stack.

Implementation Detail

Data stack stack has one element (i.e. a data stack) in it initially, whereas the data stack has no elements whatsoever.

Data stack stack is implemented by module `Nstack`, which provides the type `'a dstack_t` (d for dynamic), and a lot of dedicated function for accessing the `dss` field in various ways.

1.3 Flags

Flags stores the essential states for the virtual machine to run. For example, `curmod` records the current module whose instructions the PC is pointing at. `dss` is the data stack stack we talked about above. `scps` stores all the the names in scope stack (`scope_t list`) for scoping.

1.4 Module table

Any functional Towel source code inevitably references (or, imports) other modules. When an `inst:import` is executing, a `module_t` value is created and based on the module name, a UID is allocated for this module for referencing it later. Then the VM uses the UID as the key to store the `module_t` value into the module table.

1.4.1 Modules

A module is a compiled Towel source code loaded into TVM, it has the following important fields:

- `id` of type `module_id_t`
- `insts` of type `line array`
- `exs` of type `(name_t, value_t) Hashtbl.t`
- `imports` of type `(module_id_t, module_id_t) Hashtbl.t`

`exs` is the exported value table of the module, mapping from a name to a value.

`imports` is the absolute module id relative to this module. In the Towel compiler, names are resolved by a 2-tuple: the name id in the module and the module id. Because when compiling, the Towel compiler has no information about other modules other than the `.e` exportation file (like the header files you use in C). So it can only label other modules by IDs relative to itself (itself being zero).

When importing a new module, the module gets a global (absolute) id, all the referencing of this module should be made through this very id. So it's vital that every module maintains a mapping from the relative module id in its instructions to the absolute module id given by TVM. See 1.1 for an intuitive illustration.

1.5 Extension table

Essentially the same thing as the module table, except that it stores extensions (of type `module Ext.TowelExtTemplate`) via `inst:load-ext`.

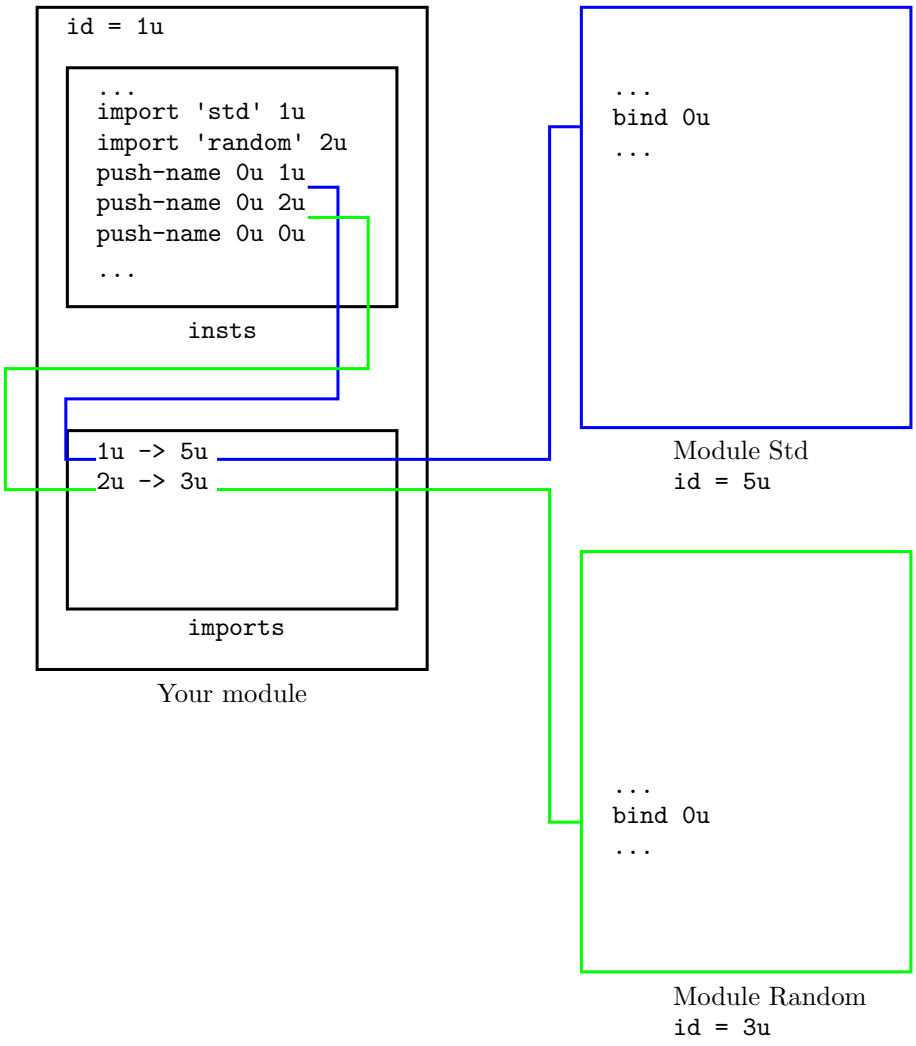

Figure 1.1: Illustration on the Usage of the `imports` table

Chapter 2

Towel Assembly Language and Its Instructions

2.1 Overview

Towel Assembly Language has the same (or less) lexical elements as the Towel programming language. And it's grammar is very simple: each line of a Towel Assembly program has no or multiple labels, an instruction, and some arguments or no argument at all according to the instruction's arity.

After assembling, labels preceding a line are eliminated, and labels as arguments are replaced with their absolute position (i.e. the line number relative to the starting of the TAL source file, of the line where they appear as line labels).

Essentially, this assembly language is considered to be the intermediate representation of compiled Towel programs. And the TVM is a facility to execute IR directly. We can always compile IR further to, for example, native code or other virtual machines like JVM.

2.1.1 Data Types

- `VString`: a string surrounded by single quote
- `VAtom`: an unsigned 64bit integer followed by an "a"
- `VUFixedInt`: an unsigned 64bit integer followed by a "u"
- `VFixedInt`: a signed 64bit integer
- `VInt`: an integer of arbitrary precision followed by an "l"
- `VFloat`: a float number

2.2 Scope-related Instructions

Hint

In the upcoming sections, we use the following convention to denote arguments of the instructions:

- `IARG_SOME` means this argument is a part of an instruction.
- `SARG_SOME` means this argument is fetched from the data stack, and it will be put back potentially modified.

  Stack argument that appears first is popped first, and so forth.

Also the term TOS and `SARG_SOME` are often interchangeable if there exists only one stack argument.

- `PARG_SOME` is the same as `SARG_SOME`, except that `PARG_SOME` is popped out from the stack forever.

Hint

Almost every nullary instruction is wrapper by an inline anonymous function (i.e. shared sequence) in the module `.w` so that they can be invoked. Because this is a very low-level module, so we prepend a dot to it, meaning to hide it from regular users.

Also because of its low-levelness, we can only code this module in the Towel Assembly Language, so a Ruby script `std_gen.rb` is used to automatically generate this primitive wrapper module.

Run `ruby std_gen.rb` and see `.w.l` and `.w.e` for detail.

2.2.1 0x01 inst:push-scope

Pushes a new scope onto the scope stack.

2.2.2 0x02 inst:pop-scope

Pops the TOS of the scope stack.

2.2.3 0x03 inst:share-scope

Does absolutely nothing. Just a place-holder to indicate that this new context shares the same scope with its parent.

2.2.4 0x04 inst:bind IARG_UID

The argument `IARG_UID` denotes the ID of the name that the TVM will bind TOS to. Then TVM does it in current scope.

2.2.5 0x05 inst:fun-arg IARG_UID

`inst:fun-arg` is a compound instruction[1]. It steals (pops) from the caller's data stack[2] and binds a reference to the value to the name indicated by the `IARG_UID` in the clone of the `closure` table of the function.

Implementation Detail

If the stack turns out to be empty, this instruction saves the name-value pairs already bound (arguments stole by previous `inst:fun-arg`s) in the copy of the closure of the current function (`curfun` of the TOS of ctxs), marks the function as partial and returns that function.

Also if the stack is empty because the TOS is a phony, it removes the phony.

Hint

Why copy the `closure` table? Think about the execution of a recursive function,

```
{ctx_current with arg1 = 5; arg2 = ?}
{ctx1 with arg1 = 3; arg2 = 4}
{ctx0 with arg1 = 1; arg2 = 2}
```

If arguments were put into the table directly, applying arg2 with, for example, 10, in `ctx_current`

[1] A compound instruction is an instruction with multiple side-effects.

[2] The second top stack of DSS, or the top stack of DSS if it's a tail recursive call.

will result in `ctx1.arg2 <- 10` and `ctx0.arg2 <- 10`. In other words, you just corrupted the activation records.

So you may ask, why not putting the arguments directly on stack, and bind the given `IARG_UID` to the index of that argument, just like what `inst:bind` does? The reason is that, if this is the case, it would be almost impossible to implement automatic partialization. Because when returning the partial applied function, the stack that saves the arguments will be destroyed, and all your arguments will be lost.

Sorry for such a complicated (and probably the only complicated) instruction.

2.3 Stack-related Instructions

2.3.1 0x10 inst:push-fun IARG_START

Makes a new function value and pushes it onto the stack without evaluating it. It takes `IARG_START` as the start position of the function to be created. The module id of the function is considered to be `curmod.id`.

2.3.2 0x11 inst:push-lnil and 0x12 inst:push-tnil

Makes a new list or tuple on top of the current data stack. TVM also puts the pointer of its content onto `ctx_t.list_make_stack` of the top most `ctx_t` value to keep track where to put new values (such as values from `inst:push-lit`, `inst:push-lnil` or `inst:push-tnil`), when TVM reaches `inst:end-list` or `inst:end-tuple`, the ref on top of `ctx_t.list_make_stack` of the top most `ctx_t` is popped.

Implementation Detail

Only selected instructions have effect on these list pointers:

- `inst:eval-and-push`
- `inst:push-lit`
- `inst:push-phony`
- `inst:push-name`
- `inst:push-fun`
- `inst:dup`
- Instructions that return some value.

2.3.3 0x13 inst:end-list and 0x14 inst:end-tuple

Ends current list or tuple construction.

2.3.4 0x15 inst:push-lit IARG_LIT

Pushes a literal `IARG_LIT` onto the stack or the list pointer on top of `ctx_t.list_make_stack`.

2.3.5 0x16 inst:push-name IARG_NID IARG_MID

Finds the value bound to the name whose ID is `IARG_NID`, in the module with the module ID being `IARG_MID`, then pushes it onto the stack.

2.3.6 0x17 inst:eval-and-push IARG_NID IARG_MID

Takes two arguments as the description of a name (same as `inst:push-name`), finds the value it is bound to. If the value is a function, then evaluates the function.

2.3.7 0x18 inst:pop PARG_POPPED

Pops the TOS of current data stack. In other words, it gets an argument from the stack, then does absolutely nothing.

2.3.8 0x19 inst:dup SARG_SOME

Duplicates the TOS of current data stack. In other words, it gets an argument from the stack, then puts back two.

2.3.9 0x1a inst:reverse PARG_N SARG_1 SARG_2 ... SARG_N (deprecated)

Takes an argument from the stack, then reverses the `PARG_N`-top most elements.

2.3.10 0x1b inst:unpack PARG_E

Assumes the TOS is a list or tuple, then unpacks it onto the stack.

Example

```
dsck: [ ... | [1 2 3]]
after unpack: [ ... | 1 | 2 | 3]
```

2.3.11 0x1c inst:pack PARG_N PARG_1 PARG_2 ... PARG_K ...

Takes one integer `PARG_N` from the stack to pack `PARG_N` items on the stack into a list. If `PARG_N` is negative one, it packs all the elements on the stack until it encounters a phony, in which case the phony will be popped.

Example

```
dsck: [ ... | 1 | 2 | 3 | 3]
after pack: [ ... | [1 2 3]]
```

Hint

If there were insufficient number of stack items, TVM throws an exception and terminates.

2.3.12 0x1d inst:push-phony

Pushes a phony onto the stack that fakes an empty stack. That is to say, whenever TVM encounters a phony, it regards the stack as being empty.

2.4 Function-related Instructions

2.4.1 0x20 inst:push-stack

Pushes a new data stack onto the data stack stack.

2.4.2 0x21 inst:share-stack

Same as `inst:share-scope`.

2.4.3 0x22 inst:pop-stack

Pops the current data stack.

2.4.4 0x23 inst:eval-tail IARG_NID IARG_MID

Finds the function the two arguments references to, then calls it in a tail recursive manner.

Implementation Detail

Some complications with the current implementation of tail recursion:

- If you use something that pushes a new scope in a tail recursion, and before this scope is popped, you tail recursed the same function again, then your scoping are leaking.

  The right way to do this is to count how many scopes are pushed during this tail call context, and before the next tail call, pop as many scopes as the counter indicates.

- Don't remember to clear the `ctx_t` field `list_make_stack`. This is likely to be ignored. However, this implies that calling a function in a tail recursive manner during a list creation will most like get you wrong results.

2.4.5 0x24 inst:ret SARG_RET_VAL

Take care of the return value of the function. Pops the execution stack. And jumps to the return address.

Implementation Detail

The actual implementation is a bit harder than this, because you have to handle the situation where a function returns its return value into a pending list or tuple, e.g. `[(1 2 +)]`.

Also note that because we're returning control to the caller, the pending list or tuple we seek is in the caller's `ctx_t`, i.e. second top most value of `ctxs`, rather than the TOS of `ctxs`.

2.4.6 0x25 inst:shared-ret

Just sets the instruction pointer back to where this shared sequence gets called. No return value copied whatsoever (because there isn't any).

2.4.7 0x26 inst:closure IARG_NID IARG_MID SARG_FUN

Assumes `SARG_FUN` is a function, takes two argument to describe the name `SARG_FUN` should capture.

Implementation Detail

Because you only need to capture names from self-module, so `IARG_MID` is often left being zero[a] when compiling.

[a] The self-module ID, i.e. the ID modules reference themselves to.

2.4.8 0x27 inst:invoke PARG_FUN

Assumes TOS is a function, then invokes it. `PARG_FUN` is popped before evaluating it.

2.4.9 0x28 inst:call IARG_START

Records $PC + 1$ as the return address, then jumps to `IARG_START`.

2.4.10 0x29 inst:install SARG_FUN

Install the captured names of `SARG_FUN` into current scope so that users can reference them.

2.4.11 0x2a inst:sweep

Sweeps the stack to get rid of all the existing elements. Used under tail recursive calls to avoid stack leakage.

Hint

This instruction is generated for every function after all the `inst:fun-arg` instructions are finished. But it's only operational when `ctx.is_tail_recursive_call` is `true`.

Hint

This instruction only jumps within the same module.

2.5 Conditional Branching Instructions

2.5.1 0x30 inst:jump IARG_POS

Unconditionally jumps to `IARG_POS`.

2.5.2 inst:j* IARG_POS SARG_X

Tests the TOS with specific predicates. If the result is true, TVM jumps to `IARG_POS`, otherwise, TVM executes the next instruction.

`inst:j*` instruction family consists of the following instructions:

- `0x32 inst:jgz`: `SARG_X` is greater than zero,
  `0x31 inst:jgez`: `SARG_X` is greater than or equal to zero
- `0x34 inst:jlz`: `SARG_X` is less than zero,
  `0x33 inst:jlez`: `SARG_X` is less than or equal to zero
- `0x35 inst:jez`: `SARG_X` is equal to zero,
  `0x36 inst:jnez`: `SARG_X` is not equal to zero
- `0x37 inst:jt`: `SARG_X` is `true`,
  `0x38 inst:jf`: `SARG_X` is `false`

2.5.3 inst:hj* IARG_POS PARG_X

Hungry versions of `inst:j*` instruction family, which consume the TOS they test against. Their `opcodes` are the ones of their non-hungry counterpart plus `0x10`. For example, the `opcode` of `hjgz` is `0x42`.

2.5.4 0x39 inst:je IARG_POS, 0x3a inst:jne IARG_POS

Both of them test if the data stack is empty. `inst:je` jumps to `IARG_POS` if it is empty, whereas `inst:jne` jumps only when the stack is non-empty.

Implementation Detail

If `inst:je` is tested against a phony, it pops it before branching.

2.6 Arithmetic Instructions

2.6.1 `0x50 inst:add, 0x51 inst:sub, 0x52 inst:mul, 0x53 inst:div`

Takes two arguments `PARG_1` and `PARG_2` from stack. Does arithmetic operations with `PARG_1` and `PARG_2`, and pushes `PARG_2 op PARG_1`. These instructions work for all the number types[3].

2.6.2 `0x55 inst:mod PARG_1 PARG_2`

Does modulo operation between integer types.

2.6.3 `0x56 inst:equ PARG_1 PARG_2`

Does equality test on `PARG_1` `PARG_2`, and puts back `true` or `false` atoms.

Hint

If two different types are tested against each other, TVM exits.

2.6.4 Bitwise Instructions

Fixed integers, unsigned fixed integers, and big integers are supported by the following bitwise arithmetic operations:

- Bitwise and, `0x57 inst:and PARG_1 PARG_2`
- Bitwise or, `0x58 inst:or PARG_1 PARG_2`
- Bitwise xor, `0x59 inst:xor PARG_1 PARG_2`
- Bitwise not, `0x5a inst:not PARG_1`
- Bitwise shift left, `0x5b inst:shl PARG_X PARG_N`
- Bitwise shift right, `0x5c inst:shr PARG_X PARG_N`
- Bitwise logical shift right, `0x5d inst:lshr PARG_X PARG_N`

Note that `inst:not`, `inst:and` and `inst:or` also work on the `true` and `false` atoms.

2.6.5 Conversion Instructions

These instructions allow one type of TOS to convert into value of other types. They are of the following form:

- `0x62 inst:to-fint PARG_X`
- `0x63 inst:to-ufint PARG_X`
- `0x64 inst:to-int PARG_X`
- `0x65 inst:to-float PARG_X`
- `0x66 inst:to-str PARG_X`

[3] Atoms are not numbers.

2.7 Instructions for Enumerables

2.7.1 0x5e inst:car PARG_XS

Does CAR operation on linked lists and strings. It takes a list or string from the stack, puts back the head of the list or string.

2.7.2 0x5f inst:cdr PARG_XS

Same as `inst:car`, except that it puts back the rest of the list or string.

2.7.3 0x60 inst:cons PARG_XS PARG_X

Does CONS operation on linked lists and strings. It constructs a new list or string depending on the type of operands and then puts it back onto the stack.

Hint

If it encounters a tuple, TVM complains and exits.

2.7.4 0x61 inst:list-empty PARG_XS

Tests if the TOS is an empty **list, tuple or string**. It puts back whether `true` or `false`, depending on the emptiness of the list, tuple or string.

2.7.5 0xf3 inst:tuple-at PARG_N PARG_XS

Pushes back the `PARG_N`-th element of `PARG_XS`.

2.8 I/O Instructions

2.8.1 0x67 inst:show PARG_X

Prints the TOS to standard output.

2.8.2 0x68 inst:read

Reads a string from standard input and store that string on the stack.

2.9 Extension Instructions

2.9.1 0xf1 inst:load-ext PARG_EXT_STR

Loads the extension by the filename of `PARG_EXT_STR` from filesystem, and records it in the `opened_exts` table. Then puts back the handle to that extension module.

2.9.2 0xf0 inst:extcall PARG_EXT_ID PARG_EXT_CN

Takes two unsigned integers from the stack. Argument `PARG_EXT_ID` is for specifying the extension module handle, the other one is the call number to indicate which routine to call from the extension.

2.10 Miscellaneous Instructions

2.10.1 0x70 inst:import PARG_MOD_STR IARG_REL_ID

Finds the module by the filename of `PARG_MOD_STR` and assign it's ID relative to current ID as `IARG_REL_ID`[4].

TVM will assign a unique ID to every module imported based on this `PARG_MOD_STR`. For example, the module string of module `Std` from the Towel standard library (or Towelib) is `'std'` (because the source filename of it is `'std.t'`).

2.10.2 0x7f inst:export IARG_UID

Exports the value bound by the name ID `IARG_UID` to the module's `exs` field.

2.10.3 0x71 inst:dint

TVM pauses execution, and invokes the debug interrupt handler, which waits for the user to input a debug command of one line and then go on executing. Currently supported commands are

- `ss`: print scope stack
- `ds`: print data stack stack
- `vi`: print the value at given stack index.
- `ln`: lookup the index the given name is bound to

See also `https://github.com/qwert42/ketivm/blob/master/vm/keti.py#L164`, which is an implementation of a similar instruction.

2.10.4 0x72 inst:type PARG_X

Pushes the type information of TOS onto the stack. This instruction only recognizes built-in types.

2.10.5 0xff inst:not-implemented

Prints an error message stating that an unimplemented instruction was found and the virtual machine has to exit.

2.10.6 0x0 inst:idle

Does absolutely nothing.

2.10.7 0xfe inst:terminate

Terminate the Towel Virtual Machine. Or when finishing importing modules, go back to original module.

Hint

Look at the similarities between (`inst:call`, `inst:invoke`, etc. `-> inst:ret`) and (`inst:import -> inst:terminate`).

[4] See previous section on modules for more details.

Chapter 3

Bytecode Form of the Towel Assembly Language

3.1 Bytecode Form Overview

A Towel Assembly Language source code can be compiled into a binary file. The layout of such a binary file is as follows:

- Magic number `0x4242`, denotes that this is a Towel binary
- The data segment, which contains all the literals used in the original source code
- Magic number `0xff3080ff`, denotes that data segment ends here
- The code segment

3.2 Data Segment

A Towel binary contains only one data segment. Data segment contains all the literals used in the original source code file, which are of type `VString`, `VAtom`, `VFixedInt`, `VUFixedInt`, `VInt`, `VFloat`.

To store a string in the data segment. First, put a tag[1] ``s'' into the data segment. After that tag, put a little endian 32-bit unsigned integer which represents the length of the string. Then put the contents of the string after the 32-bit integer.

To store an atom, put a tag ``a'' and then an unsigned little endian 64-bit integer.

To store a fixed integer, put a tag ``f'', then a signed little endian 64-bit integer.

An unsigned fixed integer is the same as an atom, except the tag is ``u''.

To store an integer (i.e. big integer) or a float, put their respective tags (``b'' for integer, ``n''[2] for float) before their binary string representations.

Example

For a float `3.14159`, the Towel literal binary representation is `6e07000000332e3134313539`.

The order of the literals are put in the data segment is their respective labels.

[1] Actually a string.

[2] "n" for not a number.

3.3 Instruction Format

Any bytecode instruction is a little-endian 64 bit number. The first byte of the number is `opcode`. For unary instructions, the second up to eighth bytes consists of the argument label. For binary instructions, the second up to fifth bytes consists of the first argument label, while the rest (sixth to eigth, 3 bytes, less than the first argument) consists of the second argument label.

When encountered a bytecode instruction, use the argument labels (if any) to retrieve actual arguments from data segment, and then form the actual instruction.

Example

Suppose we have the following bytecode file:

```
BB
  s03000000std
  u0100000000000000
  u0200000000000000
  s12000000Hello, world
  u1500000000000000
ff3080ff

1501000000000000
7002000000000000
1703000000050000
1504000000000000
fe00000000000000
```

First, we can identify this file as a Towel bytecode because the first two bytes are `0x4242`, i.e. `BB`.

We also know that there are 5 literals in the data segment:

1. a string: `'std'`
2. an unsigned fixed integer 1
3. an unsigned fixed integer 2
4. a string: `'Hello, world'`
5. an unsigned fixed integer 0x15

Then, for the first instruction (8 bytes), we get `opcode` from the first byte, in this case, `0x15`, which is `inst:push-lit` with `IARG_LIT`, so we reinterpret from the 2nd to the 8th byte as an integer, 1. And get an element from the data segment by the index of one, which turns out to be the `'std'` string.[a]

So we can finally reconstruct the first instruction to be `PUSH_LIT('str')`.

Now let's look at the third instruction, we lookup the `opcode 0x17` is `inst:eval-and-push` with `IARG_NID` and `IARG_MID`, so we reinterpret from the 2nd to the 5th byte as an integer, i.e. 3, and 6th to 8th as another integer, 5.

Thus the two arguments are `2` and `0x15`. So we can also reconstruct this instruction to be `EVAL_AND_PUSH(2, 0x15)`. The rest instructions are the same.

[a] Yes, the index of data segment starts from one.

Implementation Detail

One of the neat things about this kind of representation is that I don't have to modify the assembler even when I have variable-sized instructions, because the addresses I'm jumping to are counted by

instructions from the start, rather than bytes.

Chapter 4

Extending the Towel Virtual Machine (OCaml Implementation)

By using the Extension interface (in `src/vm/ext.ml`), you can easily add new functionalities to TVM, enabling more powerful Towel programming experience.

4.1 Prerequisite

The library (no matter if it's third-party, or of your own) should be **OCaml-compliant**, because TVM is built on top of OCaml too. That is to say, you should be able to write OCaml programs with it, thus TVM will have access to it via dynamic linking.

Hint

I think plain C libraries with only thin OCaml wrappers (no `ml` files) would work too. Because `Dynlink` only needs it to be a OCaml-aware shared-library.

4.2 Approach

Two instructions were implemented in TVM: `inst:load-ext` and `inst:extcall`.

`inst:load-ext` takes an `ext_str` from the stack and loads the corresponding extension module into TVM via the OCaml standard module `Dynlink`. An `ext_str` is a filename of the OCaml object file. After successfully loading it, the instruction leaves an unsigned integer (`OVUFixedInt`) on the stack for future reference of the extension.

Hint

`ext_str` should always ends with `.cmo` even if the target is a `.cmxs` file.

`inst:extcall` takes two arguments from the stack: it first takes the TOS as the extension handle (the one `inst:load-ext` pushed), then it takes another unsigned integer as the extension function call number (although this integer will be converted into an OCaml `int`, but I think 2^{31} is fairly enough for ordinary libraries). The extension module then matches against the call number to determine which action to perform on the data stack.

4.3 Howto

Hint

The following step illustrates how to build native extension rather than bytecode extension. If you want bytecode extension, use `ocamlc` instead, and probably lose some of the parameters.

The first step, create a OCaml plugin. This plugin file is essentially a dynamic library (e.g. a `.so` file on *nix, a `.dll` file on Windows). When implementing this plugin, you must implement the module interface `TowelExtTemplate` laid out in `src/vm/ext.ml`. It's simple: one function - `extcall`. You will have to route individual call numbers to different routines using this function.

The first argument of `extcall` is the call number, then the second one is the `dss` of the VM, for you to get specific arguments from the data stack or leave your results on it. But be careful, if you ruin this dss, the TVM is doomed.

After you finish your `extcall` function and other local functions, remember to register your module value in the `__ext__` slot so that when loading this extension, TVM will get to know of your extension.

The second step, you compile your source code via the following command:

Example

```
ocamlfind ocamlopt -package %1 %2 -shared -linkall -I %3 -o %4
```

Of these parameters:

- (`%1`) the packages your extension module uses, take `ext_random.ml` as an example, `%1` will be replaced by stdint;
- (`%2`) the extension module source file, `ext_random.ml`, for example;
- (`%3`) the directory where `t.cmi` and `nstack.cmi` are;
- (`%4`) the output binary filename, `ext_random.cmxs`, for exmpale;
- If you use some libraries that are not included in standard library, you may want to use `-linkpkg` instead of `-linkall` (for example, when building binding library for Tk).

The command for compiling the Random extension module is like this:

Example

```
ocamlfind ocamlopt -package stdint ext_random.ml -shared -linkall -I ../../build/src/vm
-o ext_random.cmxs
```

Third, place the `.cmxs` file in the `libpaths` of TVM, default locations are current working directory and the folder `towelib`. Modify `src/vm/config.ml` to change these locations.

Last, write a Towel wrapper module for your extension module. In the `.w` instruction wrapper module, name `!>ext` is bound to the `inst:load-ext` instruction wrapper and name `!>>` for `inst:extcall`. You should always bind a name to the extension handle pushed by `!>ext`.

Hint

It is often a good idea to bind a local partial applied function with the handle already applied to `!>>`.

Part III

The Implementation of the Project[1]

[1]Special thanks to Professor Edwards for so many valuable suggestions.

Chapter 1

Project Plan

1.1 A Little Bit of History about Towel

The original idea of Towel was to create an esoteric programming language like *Shakespear* or *Piet*. The central idea was that every program structure is identified by the first character. For example, `if is is-not` are all recognized as an `IF` token. With this tokenizer, you could write a GCD program in a lorem-ipsum manner.

But this soon turned out to be infeasible when you want to name more than 26 things. So the one-character grammar idea got abandoned.

But still, I want to create something different, especially in the way the programs will look like. So I thought to myself, we already have the Lisp family (prefix), as well as the ML family (infix). Let's do postfix![1] Soon the idea of stack-based postfix-syntaxed functional language came into being. Then I named it after my favorite novel series.

1.2 Specification and Development Process

Specification was not clear at the beginning of the development, but became clearer and clearer as the development went on.

The three of us get together every Monday afternoon before meeting with Professor Edwards. We would discuss the newly implemented parts and ideas, and hoped this discussion would clearify things for them as well as myself, and identify as much flaws as possible. Then during the week, new parts would get developed, and flaws fixed.

Later in the development (after the test suite was implemented as well as several test cases), the development process began to be like Test-Driven Development spontaneously.

Manuals (or specification) were modified along development (at least for the first month) to fit what we had at that time. Later in the development, because features got added or redone too fast, they quickly went out of date, and finally got a major revision after feature freeze.

1.3 Testing Process

After each major feature was completed, one or two test cases would be created to test if the feature would work.

Not long before feature freeze, some holistic test cases were made to ensure everything works correctly together. Such test cases include, for example, `quicksort.t` for quicksort and `adv-obj.t` for advanced OO simulation. It was during these tests, some serious defects were discovered, for example, scope leakage due to unbalanced `inst:pop-scope` instruction during code generation for sequences[2], or improperly designed

[1] At that time, I didn't realize that there are so many postfix languages already!

[2] Fixed in commit `adb1d3c752`.

data structure in the virtual machine that causes function calls in list literals to return the return value to wrong location[3].

After the feature freeze, tests are still being added to the test suite.

1.4 Programming Style

Basically, we used Unix-style naming conventions, i.e. underscores.

The whole indentation rule we used was the default configuration of `ocp-indent`.

Particularly,

- No lines should exceed 80 characters wide.
- Append `_t` to most type names.
- `in` clause starts at the second line of that `let-in` expression.
- `then` and `else` clause both starts a new line, i.e.

```
if something
then some_other_thing
else something_else
```

- Prefer `begin` and `end` over parentheses.
- Modularize the source files: the functions contained in one OCaml file should only serve one purpose. Except some modules, such as `Common`.
- `let`, `try`, `if`, and `match` starts right after `in`, `then`, or `else`:

```
let x = Some(1)
in if something
  then try if something
            then ahh
            else ohh
       with Error ->
         raise Failure
  else match a with
      Some(x) -> x
    | None -> raise NothingHere
```

1.5 Project Timeline

We had no designated project timeline whatsoever. By the time I consider the language to have become stable[4], it was about just right about two weeks before the report due. We got lucky.

Here's the major changes made in the code repository:

- Sep 24: THE Initial Commit.
- Sep 27: Proposal commited.
- Oct 9: Somewhat working scanner and parser.
- Oct 20: LRM commited.
- Nov 8: Got a working compiler.

[3] Fixed in commit `ccdf962de7`.

[4] Although I fixed another several severe bugs later.

- Nov 15: Hello world!
- Nov 17: Towel standard library commited.
- Nov 29: Major change to the VM.
- Dec 3: Extension mechanism added to the VM.
- Dec 4: The language become dynamic officially (dynatowel branch merged into master).
- Dec 6: Bytetowel merged.
- Dec 11: Feature freeze.

1.6 Roles and Responsibilities

- Project Manager: None
- The Guru Guy: Zihang Chen
- System Architect: Zihang Chen
- Testers: Zihang Chen, Baochan Zheng, Guanlin Chen

1.7 Tools and Languages Used in Developemnt

- OCaml 4.02.
- `opam` for installing OCaml packages, `findlib` for compiling my source code with the packages, and `ocamldebug` for some information on stack trace. Overall package usage: `Batteries`, `Stdint`, `Sha`, `Extlib`, `menhir`.
- Emacs 24.5 with `tuareg-mode` (OCaml syntax highlighting), `merlin` (syntax and semantics checking, linting) and `ocp-indent` (indenting OCaml). These are recommended by the `opam` official website.
- Ruby 2.2.3 is used for test automation and compiler/VM source code generation (to write as less boilerplate code as possible).
- Waf (a Python3 tool) is used as the building framework of the project, Python3 is used for the building script.
- XeTeX for document compilation and AUCTeX for LaTeX editing.
- Git for project version control, GitHub for project hosting, and Google for some of the problem solving. English for understanding the OCaml and Batteries manual, also the solutions I found on Stackoverflow.

1.8 Project Log

See Appendix A.

Chapter 2

Architectural Design

2.1 Overall

In this chapter, we consider the term intermediate representation to be synonymous with the term Towel Assembly.

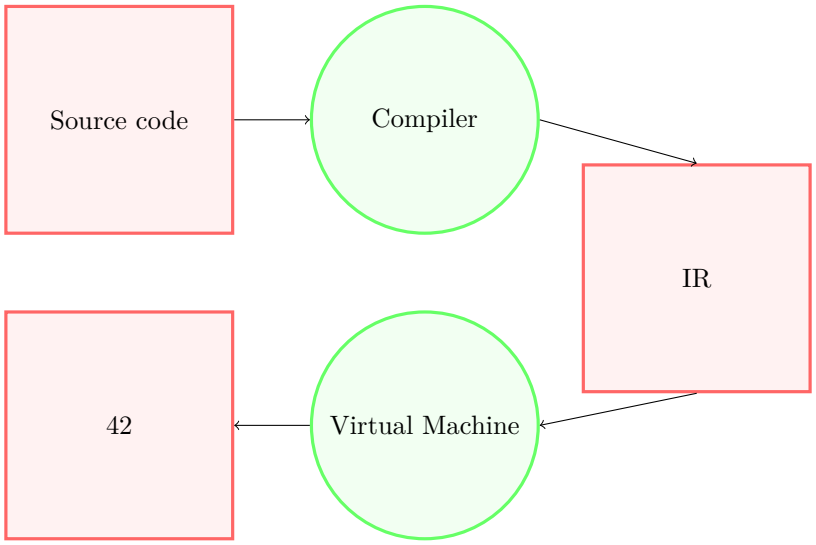

Figure 2.1: Overall Architecture

2.2 Structural Overview on the Towel Compiler (Zihang Chen)

The compiler frontend (module `Weave`) first invokes `Switches.parse` to parse the switch configuration for the source code. Then it gets the abstract syntax tree out of the source code by tokenizing and parsing it with modules `Scanner.token` and `Parser.sentence`.

After that, it invokes module `Compile.compile` to transform the AST data structure to the AST of intermediate representation. While traversing the AST, module `Compile` also makes use of module `Scoping` to do scope analysis and report any names that get referenced to without first binding something to it. If such name is found, the compilation aborts immediately.

Implementation Detail

Module `Cseg` defines an infix operator `(|~~|)` to represent the concatenation of two `Tasm_ast.asm` data

structures into one. This greatly simplifies the work for code generation. Because you can write the code generation part in a really straight-forward way, like this:

```
let snippet = cnil
              |~~| preamble
              |~~| (line INSTALL)
              |~~| fun_args
              |~~| (line SWEEP)
              |~~| body_inst
              |~~| (cline [end_label] (Some(POP_SCOPE)))
              |~~| (line RET)
              |~~| (put_label [real_end_label])
```

The `Compile.compile` function basically takes a `Sentence`, then for each word in the `Sentence`, it invokes `g_word`. For each kind of word, `g_word` route them to respective `g_*` functions. For each kind of word, there exists a corresponding `g_*` function.

Each `g_*` function takes a parameter `ctx` to keep track of the current context of the code generation, for example, the scope environment the piece of AST is in, or if the piece of AST is backquoted, etc.

Some of the `g_*` functions (specifically, those words that represents a single value, for example, literals, sequences, etc) also take another parameter `inst_ctx` for the `g_bind` function to generate proper `inst:bind` instruction.

With the IR AST, the compiler invokes `Assemble.assemble` to get rid of all the labels, and replace them with absolute positions. After this step, the compiler gets Assembled IR AST.

In the last step, the compiler invokes `Tasm_bytecode.b_asm` to procure the bytecode representation of the IR.

Hint

The above mainly talked about the very common invocation to the compiler, for other uses, for example, outputing text version of Assembled IR AST, the compiler uses `Tasm_stringify.p_asm`.

See also Figure 2.2.

2.3 Structural Overview on the Towel Virtual Machine (Zihang Chen)

Things are much simpler on the VM-side. The virtual machine frontend `Tvm` first reads in the bytecode representation of IR, transforms it back into IR AST by invoking `Tasm_inv_bytecode.parse_bytecode`. Then invokes `Nvm.exec` to execute all the instructions parsed. `Nvm.exec` is a simple wrapper function to bootstrap `Nvm.__exec`, which actually does all the work.

The function `Nvm.__exec` implements all the instructions available in a very big `match` expression. After each execution of an instruction, it tail calls itself with the new state of the virtual machine (for example, `flags`, IP, etc.) to execute the next instruction pointed by IP. For more on the state of the Towel Virtual Machine, see also chapter 1.

For certain instructions that are quite alike, for example, the arithmetic instructions and branching instructions, we use Ruby scripts to generate respective code for them, then invoke them in `Nvm.exec`.[1]

Implementation Detail

The `Nvm.exec` depends on many other functions from other modules to work. For example,

- The module `T` provides all the basic data types for the TVM, an example would be the value type `T.value_t`.

[1] An alternative to do this is by `camlp4`.

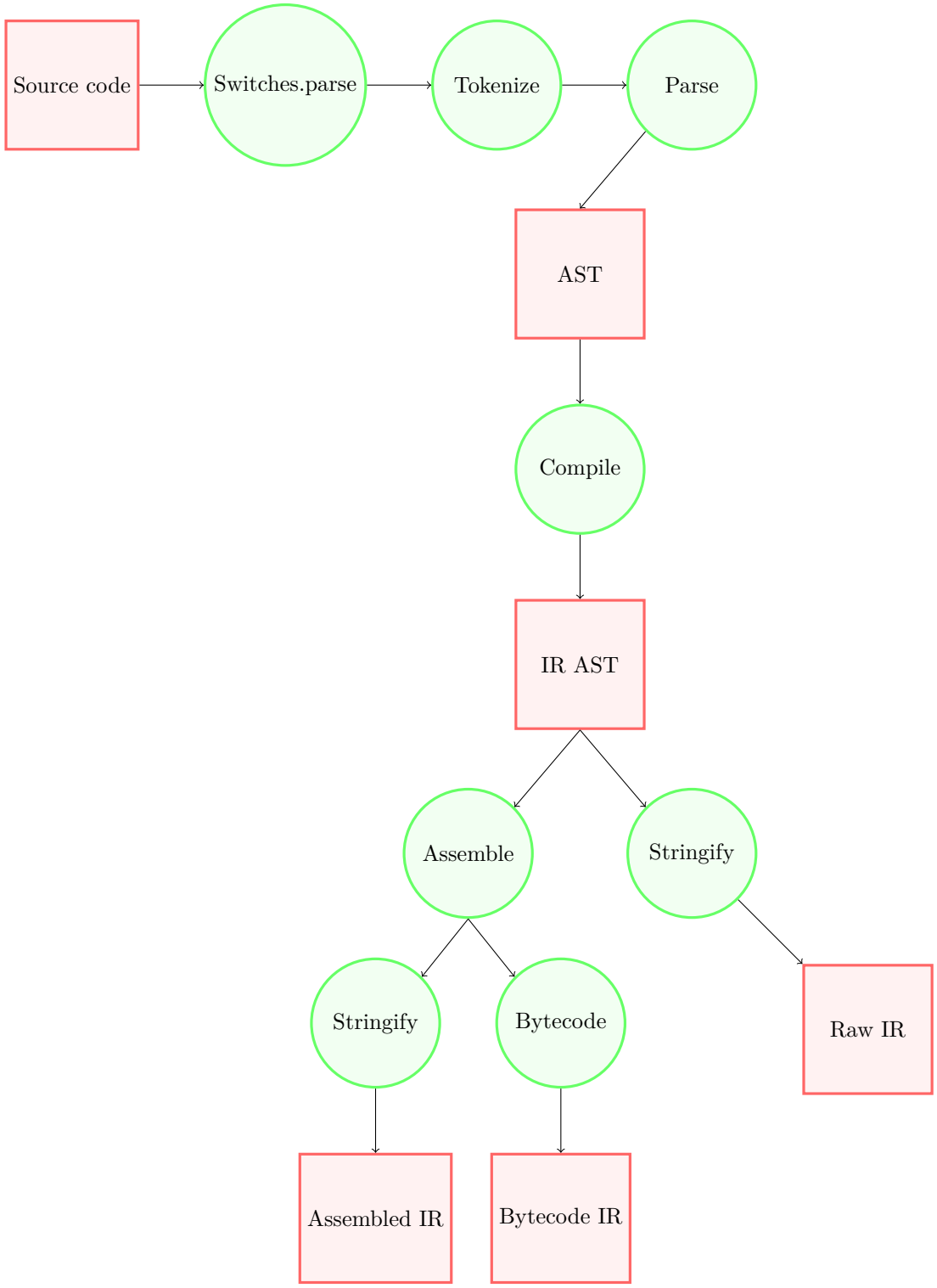

Figure 2.2: Compiler Architecture

- The module `Vm_t` provides various data types specific for this implementation of the TVM, such as `Vm_t.flags_t`.
- The module `Nstack` provides support for the stack data structure and various operations used by `Nvm.exec`.
- The module `Arithmetics` provides implementation for various arithmetics instructions.
- The module `Jumps` provides `Jumps.branch` to do branching.
- The module `Dscoping` provides data structure and operations for scoping.
- The module `Ext` implements extension (plugin) mechanism for `inst:load-ext` and `inst:extcall`.

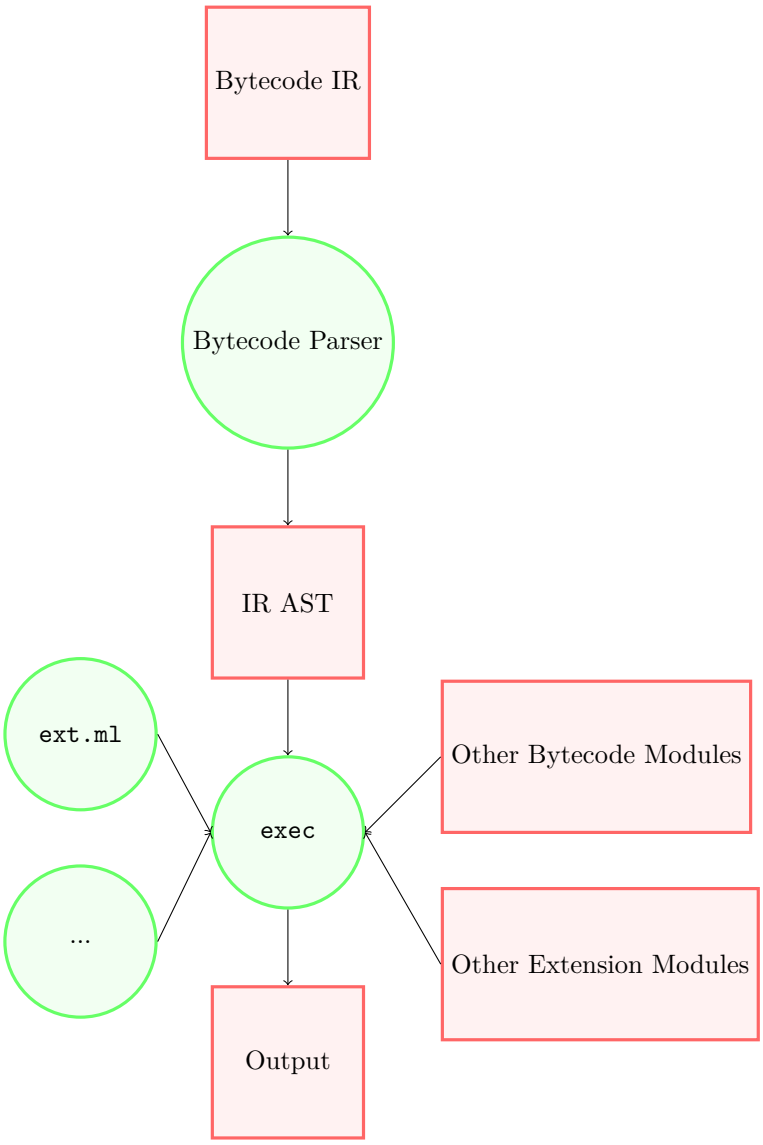

Figure 2.3: Virtual Machine Architecture

2.4 The Towel Assembly Components (Zihang Chen)

The script `ccg.rb` generates a whole bunch of Towel Assembly related modules for the compiler or the virtual machine to generate bytecode or parse text form IR.

The generated modules include:

- `tasm_ast.mli`

  defines abstract syntax tree types

- `tasm_scanner.mll`

  scanner for text form IR

- `tasm_parser.mly`

  parser for text form IR

- `tasm_stringify.ml`

  provides functions for converting AST to human-readable string

- `tasm_bytecode.ml`

  provides functions for converting AST to VM-readable bytecode

- `tasm_inv_bytecode.ml`

  provides functions for parsing bytecode back to AST

Implementation Detail

The `ccg.rb` script reads information on the instructions from the accompanied Ruby script `inst.rb`. So when tweaking instructions for the Towel Assembly Language, only `inst.rb` is changed.

Chapter 3

Test Plan

3.1 Individual Test Examples

3.1.1 `partial.t`

Code Listing

```
"---
Tests partial function application.
!>>
!<< 42
41
[2 2]
"

import 'std' \

bind Dec1 (1 -\Std)
then (43 Dec1 !println\Std 42 Dec1 !println\Std)
```

The raw text form of the IR is as follows:

Code Listing

```
  push-stack
  push-scope
  push-lit 'std'
  import 1u
  push-scope
  call :6e59cf543-1u-st
  bind 1u
  push-lit 43
  eval-and-push 1u 0u
  eval-and-push 25u 1u
  push-lit 42
  eval-and-push 1u 0u
  eval-and-push 25u 1u
  pop-scope
  terminate
:6e59cf543-1u-st push-stack
```

```
  push-lit 1
  eval-and-push 4u 1u
:6e59cf543-1u-end ret
:6e59cf543-1u-real-end
```

Raw text form IR are trivial. We won't show more of these.

The assembled text form of the IR is as follows:

Code Listing

```
push-stack
push-scope
push-lit 'std'
import 1u
push-scope
call 15u
bind 1u
push-lit 43
eval-and-push 1u 0u
eval-and-push 25u 1u
push-lit 42
eval-and-push 1u 0u
eval-and-push 25u 1u
pop-scope
terminate
push-stack
push-lit 1
eval-and-push 4u 1u
ret
```

The assembled binary form of the IR is as follows:

Code Listing

```
4242 7303 0000 0073 7464 7501 0000 0000
0000 0075 0f00 0000 0000 0000 662b 0000
0000 0000 0075 0000 0000 0000 0000 7519
0000 0000 0000 0066 2a00 0000 0000 0000
6601 0000 0000 0000 0075 0400 0000 0000
0000 ff30 80ff 2000 0000 0000 0000 0100
0000 0000 0000 1501 0000 0000 0000 7002
0000 0000 0000 0100 0000 0000 0000 2803
0000 0000 0000 0402 0000 0000 0000 1504
0000 0000 0000 1702 0000 0005 0000 1706
0000 0002 0000 1507 0000 0000 0000 1702
0000 0005 0000 1706 0000 0002 0000 0200
0000 0000 0000 fe00 0000 0000 0000 2000
0000 0000 0000 1508 0000 0000 0000 1709
0000 0002 0000 2400 0000 0000 0000
```

3.1.2 `bind.t`

Code Listing

```
"---
Tests if nested bind and also works.
```

```
!>>
!<< 3
-1
5
3
10
-2
9
3
12
11
"

import 'std' \

bind A fun` B,
  bind C fun` D,
    bind E (1 2 -\Std)
    then (((E !println\Std D !println\Std B !println\Std 10)))
  also F fun` D,
    bind E (5 7 -\Std)
    then (E !println\Std D !println\Std B !println\Std 12)
  then (B !println\Std 5 C !println\Std 9 F !println\Std 11)
then (3 A !println\Std)
```

The assembled text form of the IR is as follows:

Code Listing

```
push-stack
push-scope
push-lit 'std'
import 1u
push-scope
push-fun 12u
bind 1u
push-lit 3
eval-and-push 1u 0u
eval-and-push 25u 1u
pop-scope
terminate
push-stack
push-scope
install
fun-arg 2u
push-scope
push-fun 57u
bind 3u
closure 2u
push-fun 35u
bind 6u
closure 2u
eval-and-push 2u 0u
eval-and-push 25u 1u
```

```
push-lit 5
eval-and-push 3u 0u
eval-and-push 25u 1u
push-lit 9
eval-and-push 6u 0u
eval-and-push 25u 1u
push-lit 11
pop-scope
pop-scope
ret
push-stack
push-scope
install
fun-arg 4u
push-scope
call 52u
bind 5u
eval-and-push 5u 0u
eval-and-push 25u 1u
eval-and-push 4u 0u
eval-and-push 25u 1u
eval-and-push 2u 0u
eval-and-push 25u 1u
push-lit 12
pop-scope
pop-scope
ret
push-stack
push-lit 5
push-lit 7
eval-and-push 4u 1u
ret
push-stack
push-scope
install
fun-arg 4u
push-scope
call 80u
bind 5u
call 68u
pop-scope
pop-scope
ret
push-stack
call 71u
ret
push-stack
eval-and-push 5u 0u
eval-and-push 25u 1u
eval-and-push 4u 0u
eval-and-push 25u 1u
eval-and-push 2u 0u
eval-and-push 25u 1u
```

```
    push-lit 10
    ret
    push-stack
    push-lit 1
    push-lit 2
    eval-and-push 4u 1u
    ret
```

The assembled binary form of the IR is as follows:

Code Listing

```
4242 7303 0000 0073 7464 7501 0000 0000
0000 0075 0c00 0000 0000 0000 6603 0000
0000 0000 0075 0000 0000 0000 0000 7519
0000 0000 0000 0075 0200 0000 0000 0000
7539 0000 0000 0000 0075 0300 0000 0000
0000 7523 0000 0000 0000 0075 0600 0000
0000 0000 6605 0000 0000 0000 0066 0900
0000 0000 0000 660b 0000 0000 0000 0075
0400 0000 0000 0000 7534 0000 0000 0000
0075 0500 0000 0000 0000 660c 0000 0000
0000 0066 0700 0000 0000 0000 7550 0000
0000 0000 0075 4400 0000 0000 0000 7547
0000 0000 0000 0066 0a00 0000 0000 0000
6601 0000 0000 0000 0066 0200 0000 0000
0000 ff30 80ff 2000 0000 0000 0000 0100
0000 0000 0000 1501 0000 0000 0000 7002
0000 0000 0000 0100 0000 0000 0000 1003
0000 0000 0000 0402 0000 0000 0000 1504
0000 0000 0000 1702 0000 0005 0000 1706
0000 0002 0000 0200 0000 0000 0000 fe00
0000 0000 0000 2000 0000 0000 0000 0100
0000 0000 0000 f200 0000 0000 0000 0507
0000 0000 0000 0100 0000 0000 0000 1008
0000 0000 0000 0409 0000 0000 0000 2607
0000 0000 0000 100a 0000 0000 0000 040b
0000 0000 0000 2607 0000 0000 0000 1707
0000 0005 0000 1706 0000 0002 0000 150c
0000 0000 0000 1709 0000 0005 0000 1706
0000 0002 0000 150d 0000 0000 0000 170b
0000 0005 0000 1706 0000 0002 0000 150e
0000 0000 0000 0200 0000 0000 0000 0200
0000 0000 0000 2400 0000 0000 0000 2000
0000 0000 0000 0100 0000 0000 0000 f200
0000 0000 0000 050f 0000 0000 0000 0100
0000 0000 0000 2810 0000 0000 0000 0411
0000 0000 0000 1711 0000 0005 0000 1706
0000 0002 0000 170f 0000 0005 0000 1706
0000 0002 0000 1707 0000 0005 0000 1706
0000 0002 0000 1512 0000 0000 0000 0200
0000 0000 0000 0200 0000 0000 0000 2400
0000 0000 0000 2000 0000 0000 0000 150c
0000 0000 0000 1513 0000 0000 0000 170f
```

```
0000 0002 0000 2400 0000 0000 0000 2000
0000 0000 0000 0100 0000 0000 0000 f200
0000 0000 0000 050f 0000 0000 0000 0100
0000 0000 0000 2814 0000 0000 0000 0411
0000 0000 0000 2815 0000 0000 0000 0200
0000 0000 0000 0200 0000 0000 0000 2400
0000 0000 0000 2000 0000 0000 0000 2816
0000 0000 0000 2400 0000 0000 0000 2000
0000 0000 0000 1711 0000 0005 0000 1706
0000 0002 0000 170f 0000 0005 0000 1706
0000 0002 0000 1707 0000 0005 0000 1706
0000 0002 0000 1517 0000 0000 0000 2400
0000 0000 0000 2000 0000 0000 0000 1518
0000 0000 0000 1519 0000 0000 0000 170f
0000 0002 0000 2400 0000 0000 0000
```

3.2 Holistic Test Example: Quicksort

Code Listing

```
"---
Holistic test: Quicksort.
!>>
!<< [1 2 3 4 5]
"

import 'std' \

bind #quicksort ,\ L,
  (L ?#empty\Std ift (!!pop\Std []), (!!pop\Std
    bind ~h (L #hd\Std)
    also ~t (L #tl\Std)
    then (~t (~h >\Std) /filter\Std #quicksort
          [~h]
          ~t (~h <=\Std) /filter\Std #quicksort
      #concat\Std #concat\Std)))
then ([5 4 3 2 1] #quicksort !println\Std)
```

The assembled text form of the IR is as follows:

Code Listing

```
push-stack
push-scope
push-lit 'std'
import 1u
push-scope
push-fun 19u
bind 1u
closure 1u
push-lnil
push-lit 5
push-lit 4
push-lit 3
```

```
push-lit 2
push-lit 1
end-list
eval-and-push 1u 0u
eval-and-push 25u 1u
pop-scope
terminate
push-stack
push-scope
install
fun-arg 2u
eval-and-push 2u 0u
eval-and-push 36u 1u
jf 30u
eval-and-push 27u 1u
push-lnil
end-list
jump 50u
eval-and-push 27u 1u
push-scope
call 64u
bind 3u
call 60u
bind 4u
eval-and-push 4u 0u
call 56u
eval-and-push 59u 1u
eval-and-push 1u 0u
push-lnil
eval-and-push 3u 0u
end-list
eval-and-push 4u 0u
call 52u
eval-and-push 59u 1u
eval-and-push 1u 0u
eval-and-push 57u 1u
eval-and-push 57u 1u
pop-scope
pop-scope
ret
push-stack
eval-and-push 3u 0u
eval-and-push 13u 1u
ret
push-stack
eval-and-push 3u 0u
eval-and-push 10u 1u
ret
push-stack
eval-and-push 2u 0u
eval-and-push 32u 1u
ret
push-stack
```

```
eval-and-push 2u 0u
eval-and-push 30u 1u
ret
```

The assembled binary form of the IR is as follows:

Code Listing

```
4242 7303 0000 0073 7464 7501 0000 0000
0000 0075 1300 0000 0000 0000 6605 0000
0000 0000 0066 0400 0000 0000 0000 6603
0000 0000 0000 0066 0200 0000 0000 0000
6601 0000 0000 0000 0075 0000 0000 0000
0000 7519 0000 0000 0000 0075 0200 0000
0000 0000 7524 0000 0000 0000 0075 1e00
0000 0000 0000 751b 0000 0000 0000 0075
3200 0000 0000 0000 7540 0000 0000 0000
0075 0300 0000 0000 0000 753c 0000 0000
0000 0075 0400 0000 0000 0000 7538 0000
0000 0000 0075 3b00 0000 0000 0000 7534
0000 0000 0000 0075 3900 0000 0000 0000
750d 0000 0000 0000 0075 0a00 0000 0000
0000 7520 0000 0000 0000 00ff 3080 ff20
0000 0000 0000 0001 0000 0000 0000 0015
0100 0000 0000 0070 0200 0000 0000 0001
0000 0000 0000 0010 0300 0000 0000 0004
0200 0000 0000 0026 0200 0000 0000 0011
0000 0000 0000 0015 0400 0000 0000 0015
0500 0000 0000 0015 0600 0000 0000 0015
0700 0000 0000 0015 0800 0000 0000 0013
0000 0000 0000 0017 0200 0000 0900 0017
0a00 0000 0200 0002 0000 0000 0000 00fe
0000 0000 0000 0020 0000 0000 0000 0001
0000 0000 0000 00f2 0000 0000 0000 0005
0b00 0000 0000 0017 0b00 0000 0900 0017
0c00 0000 0200 0038 0d00 0000 0000 0017
0e00 0000 0200 0011 0000 0000 0000 0013
0000 0000 0000 0030 0f00 0000 0000 0017
0e00 0000 0200 0001 0000 0000 0000 0028
1000 0000 0000 0004 1100 0000 0000 0028
1200 0000 0000 0004 1300 0000 0000 0017
1300 0000 0900 0028 1400 0000 0000 0017
1500 0000 0200 0017 0200 0000 0900 0011
0000 0000 0000 0017 1100 0000 0900 0013
0000 0000 0000 0017 1300 0000 0900 0028
1600 0000 0000 0017 1500 0000 0200 0017
0200 0000 0900 0017 1700 0000 0200 0017
1700 0000 0200 0002 0000 0000 0000 0002
0000 0000 0000 0024 0000 0000 0000 0020
0000 0000 0000 0017 1100 0000 0900 0017
1800 0000 0200 0024 0000 0000 0000 0020
0000 0000 0000 0017 1100 0000 0900 0017
1900 0000 0200 0024 0000 0000 0000 0020
0000 0000 0000 0017 0b00 0000 0900 0017
```

```
1a00 0000 0200 0024 0000 0000 0000 0020
0000 0000 0000 0017 0b00 0000 0900 0017
0d00 0000 0200 0024 0000 0000 0000 00
```

3.3 The Test Suite Script: Prefect (Zihang Chen)

When Prefect is invoked, it automatically searches the current working directory for files with extension `.t`. For each `.t` file, if the filename starts with two underscores, it is simply ignored.

The test input and expected output are all embedded in the comments of their respective test case files:

Implementation Detail

```
"Some optional background or rationale description of this test case.
---
Mandatory test case description.
!>> optional test input
!<< optional expected output"
```

Again, we are using the exclamation mark to denote the side-effected outer world.

A test is said to be passed if and only if it exits normally and prints to the standard output exactly the expected output.

Code Listing

```ruby
#! /usr/bin/env ruby

require 'open3'
require 'colorize'

class Prefect
  # This class tests per file.
  IN_PREFIX = '!>> ?'
  OUT_PREFIX = '!<< ?'

  PAT = /.*".*---\n(.*)\n#{IN_PREFIX}([^"]*)\n#{OUT_PREFIX}([^"]*)"/m
  # This means that a testcase description consists of three parts:
  # (1) the text description part that tells what this testcase is about
  # (2) the input part, led by !>>\space
  # (3) the expected output part, led by !<<\space

  def initialize(fp, **ctx)
    @ctx = ctx
    @fp = fp.chop.chop

    open(fp) {|f|
      @content = f.read
    }

    extract_info
  end

  def extract_info
    matches = PAT.match(@content)
    unless matches.nil?
```

```
      @description, @input, @expected_output = matches.captures
      @description.strip!
    end
  end

  def execute_ok(idx)
    pretty_input = if @input == ''
                     'None'.light_cyan
                   else
                     @input.white
                   end
    pretty_output = if @expected_output == ''
                      'None'.light_cyan
                    else
                      @expected_output.white
                    end

    puts "Testcase #{('#' + idx.to_s).blue}: #{@description}
---
Input: #{pretty_input}
Output: #{pretty_output}"

    out, err, status = Open3.capture3(@ctx[:weave_path],
                                      "#{@fp}.t",
                                      '-o',
                                      "temp/#{@fp}.w")

    if status.exitstatus != 0
      [false, out]
      return
    end

    if ARGV[0] == '-t'
      puts "Tracing #{@fp}.w =>"
      output, _ = Open3.capture2(
          @ctx[:tvm_path], "temp/#{@fp}.w", '-t', :stdin_data=>@input
      )
    else
      output, err, status = Open3.capture3(
          @ctx[:tvm_path], "temp/#{@fp}.w", :stdin_data=>@input
      )
    end

    if status.exitstatus != 0
      [false, err]
      return
    end

    if output == @expected_output
      [true, output]
    else
      [false, output]
    end
```

```
  end
end

if __FILE__ == $0
  require 'fileutils'
  FileUtils.remove_entry_secure 'temp'
  FileUtils.mkdir 'temp'

  SUFFIX = 't'
  WEAVE_PATH = '../build/src/compiler/weave'
  TVM_PATH = '../build/src/vm/tvm'
  make_prefect = lambda {|fp| Prefect.new(
    fp,
    :weave_path=>WEAVE_PATH,
    :tvm_path=>TVM_PATH
  )}

  cnt = 0
  results = {}
  Dir.foreach '.' do |entry|
    if entry =~ /[^.]*\.#{SUFFIX}/
      if entry.start_with? '__'
        next
      end

      cnt += 1
      ok, out = make_prefect.call(entry).execute_ok cnt

      puts "Real output: #{out}\n"
      if ok
        results[entry] = :ok
        puts "#{entry} =====> #{'[OK]'.green}"
      else
        results[entry] = :failed
        puts "#{entry} =====> #{'[FAILED]'.red}"
      end

      puts '-' * 42
    end
  end

  count = results.inject({:ok => 0, :failed => 0}) do |acc, kv|
    _, v = kv
    acc[v] += 1
    acc
  end

  if count != {}
    if count[:ok] > 0
      ok = count[:ok].to_s.green
    else
      ok = 0.to_s
    end
```

```
    if count[:failed] > 0
      failed = count[:failed].to_s.red
    else
      failed = 0.to_s
    end
    puts "Summary: #{ok} passed, #{failed} failed."
  end
end
```

3.4 Tests Overview (Zihang Chen)

16 tests were included in the test suite.

- `bind.t`

  tests whether nested `bind` forms and `also` clauses work or not.

- `functional.t`

  tests some of the functional features of Towel, for example, partial function application.

- `gcd.t`

  tests if tail recursion and chained `if` forms work.

- `idle.t`

  tests if `.idle` works.

- `ift-iff.t`

  tests whether `ife`, `ifne`, `ift`, `iff` and phonies work.

- `import.t`

  tests whether implicit import and explicit import work as expected.

- `list.t`

  tests various functions for enumerables, for example, `#hd\Std`, `#1\Std`, etc.

- `macro.t`

  tests if shared sequences work, also tests the expressiveness of Towel.

- `obj.t`

  mimic an object-oriented programming situation, intended as testing for function closures.

- `partial.t`

  tests if partial function application is stable.

- `phony.t`

  tests if phony, partial function application and `!invoke\Std` can work together.

- `quicksort.t`

  a holistic test, tests if non-tail recursion, the stability (whether something leaks or not) of `bind-then` forms in non-tail recursive environment, list functions, and the overall stability of the virtual machine.

- `rnd.t`

  tests if the extension mechanism of the virtual machine works.

- `test.t`

  the hello world test.

- `zip.t`

  tests if complicated list creation works (especially, function calls inside list literals) and the stability of tail recursions.

Chapter 4

Lessons Learned

4.1 Zihang Chen

First of all, OCaml is great (but not perfect). Be patient and learn how to program in it. It will change your life (for someone who hasn't been doing much functional programming).

Discuss your ideas with your teammates, explain your code to people, fix your bugs right away. Keep a memo at hand, even before sleep.

If you cannot explain your idea or code to others, you don't understand them at all.

Don't be afraid to rewrite 90% of your code. Make sure every time you do this, you reduce 10% of the code.

You'll never get to fix all your bugs.

4.1.1 Some Practical Advices

1. RTFM!! It is hard to understand even the words in them at first, but you'll get to the gist after reading them for 10 times.

2. Code style matters.

3. Remember to quote your data type constructor and its arguments with a pair of parentheses when working with function application. The OCaml compiler is really stupid about this: `some_function Some(x)`. The right way to do it is `some_function (Some x)`, or `some_function (Some(x))`. There are many other traps like this that will ruin a whole day for you, so watch out!

4. Use GitHub to host your code, enjoy the little deep green squares in your Contributions chart.

5. Use `opam`, `Batteries` and `Findlib`. Learn to program OCaml in Emacs, or at least Vim. Never program in Sublime Text.

6. When in trouble, first use `ocamldebug` to see the backtrace and identify the line number where your program blows up. Then use print to debug. (And maybe the time travel feature of `ocamldebug`.)[1]

7. Always wrap `Hashtbl.find` in a `try` expression, because it's really frustrating to see only a `Failure: Not_found` printed when your program runs into trouble.

8. Don't panic.

[1] Because OCaml loses most of the type information after compilation, you can't really get something useful out of the debugger.

4.2 Baochan Zheng

Working on this unique project with a smart student is really challenging.

The most important thing I learnt this semester is that communication is extremely important for a teamwork task. From the very beginning, we should have argued more about our project and make it more achievable for us. Besides, although we stick to meeting at least once a week, we are so used to listening to Zihang's idea and doing things he told us to, but never challenge him. This bad habit makes us took lots of detours, for example we've been to "data stack stack stack" things and wasted a lot of time on "object reference table" which is proved to be an inefficient way.

On the other hand, although we spent almost the whole semester in struggling to follow Zihang's crazy idea, we do learn a lot from him. He showed us how to make a language general-purpose, how to make it efficient and elegant. And we tried to build our own virtual machine (although our coding on virtual machine was updated by Zihang after a essential change on basic structure), which is the thing many other groups are not working on. What's more, we even learnt the basic rules of Ruby to finish the module coding. We gained deep understanding on runtime environment, stack management and other things mentioned in class.

Losing control of my own project doesn't feel good, but I'm still happy about having this chance to work with Zihang, and engaging in this crazy project.

4.3 Guanlin Chen

From the whole semester, under the tutoring of PLT class, we implemented what we learned and brought the knowledge into the real world. Towel language is a stack based programming language which roots in our team leader Zihang's mind. The logic of the programming language is complicated and requires comprehensively analysis, neatly blueprint, also intelligence.

Together with Baochan, my work focuses on the virtual machine part. Even dealing with a minor part of the whole language , I am sometimes struggled to figure out the logic and the method to construct the VM. Impressively, the 'four stack' ages was the most challenging days, CTXS, DSS, SCPS and ORT were resemble to four distinct resource station, only a properly combination could figure the desired output. Even ORT is rejected at last because of our task to simplize the language, that part of work is also a memory segment which embedded deeply in my heart. For instance, the DSS stores the reference , which is the key of the harsh table, pointing at the data. During the programming, I always mistook the real value with the reference, which gave a inproper output.

So always be logical thinking and be preoccupied when implementing something in your intricate logic. So as Zihang mentioned, 'Everytime your program confuses you, check whether the type could match.'

Another significant portion for the VM is the introduction of ruby. The basic notion occurs during the optimation of the VM. When dealing with the AST given by parser, huge amount of instructions are required, however, some of them have similar function and less distinguished programming methods. Ruby is resemble to a glue which could build script for our program to make some optimization. For the mathematics instructions portion of Towel, we should implement different mathematics operations, 'plus', 'substract', 'multiple', 'divide' for the parameters with the types like 'int', 'fint' etc...From the original method, we should deal with each of the operator with each of the type, that is time consuming and mechanical repetition tasks once some parts should be modified, it is trivial to modify all the program. In ruby, a 'map' function could be employed to "glue" different parts by visiting all the elements from a list. This method is introduced to 'math', 'to', 'jmp'related instructions.

Part IV

The Future

Although the project has accomplished most of the goals it set at the very beginning, there's still much that's left to be improved or done. For example, a native compiler that compile the intermediate representation to C code, and then procure a native application using `gcc` or `clang`.

What's more challenging, is to design a statically typed Towel. Static-typed stack-based language are rare because of their dynamic nature: a function's effect on the stack (or to put it in a StrongForth term, stack-effect) is generally not known until it is executed. So statically typing Towel will require information we don't usually define in value-based languages, namely, how many elements will be pushed (or popped, depending on the semantics of the function) after it finishes, and all their types. Type inferring is possible as well, but I suspect that the Hindly-Milner type system won't work here without any modification.

Other from the ideas above, currently, compilation error messages on the compiler-side are still very vague. It may be necessary to rewrite the compiler the fourth time to plug in line number tracing or something like that. Error handling on the VM-side is also considered very weak. Adding some kind of error recovery mechanism seems useful.

Better debugging facilities need to be implemented. Unlike OCaml, Towel is a dynamic language, there are still much information to use for debugging purposes at runtime.

There are also still large parts missing in the standard library, for example, file I/O, mutable data structure (arrays, etc.), string manipulation functions, regular expression, networking, and so on.

Appendix A

Project Log [1]

```
commit 6b14ac3fb7e502749f1b4427ddd5d48f4d647b66
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Thu Sep 24 09:30:45 2015 -0400

    Initial commit

commit be3c048d649099ceb8fdc0c0a65d2e3e089086af
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Sun Sep 27 01:16:57 2015 -0400

    proposal 1

commit 561f4165f5af9d831ad91b74b4267a39a70ce068
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Wed Oct 7 17:23:45 2015 -0400

    first stage of lexer and parser and ast types

commit d3999c3048c620333be23f4420f7aaa19147d422
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Thu Oct 8 00:18:40 2015 -0400

    compilable version, which unfortunately wont work, but debuggable with menhir

commit cf6b0de9ceacab07d9fcf23814f4e479cd32a5a3
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Thu Oct 8 09:28:02 2015 -0400

    compact ast, removed most of the `list(word)`

commit c04f7a82d120571264f9bb1b1be5a5c68c415029
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Thu Oct 8 09:37:55 2015 -0400

    temporarily stash

commit 28c19f77f610756727428fe43774456abc1e6cd1
Author: Zihang Chen <chsc4698@gmail.com>
```

[1] Up to only feature freeze. Bug-fix commits may not have been included.

```
Date:   Thu Oct 8 13:22:20 2015 -0400

    reduce AT and add BQUOTE backquote

commit b0f3c4255d355dea0b1dbb25e5a3fc958a80177f
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Thu Oct 8 13:26:02 2015 -0400

    add BQUOTE backquote

commit 0bc4304992fecb3927b2db6b096f3f280e2d5cc4
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Fri Oct 9 20:53:56 2015 -0400

    no warnings

commit 68bfc48a5b2875c9a255f95058c684b3d3737255
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Fri Oct 9 20:57:53 2015 -0400

    enhance namespace resolving

commit 10f3f091ddfac20e885cb90d8271e51dd9e921a1
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Sat Oct 10 02:09:38 2015 -0400

    somewhat working parser and lexer

commit 4ca60339a2c2aad4e757b5bed40e4ec649c891e7
Merge: 28c19f7 10f3f09
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Sat Oct 10 02:16:08 2015 -0400

    merge from compact-ast

commit 1594b9903a7c6dc002357569a113f2ed22bbfdd5
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Sat Oct 10 02:42:03 2015 -0400

    redesign type definition

commit a4ae88601b412909d1074eb47dc002f6eb4a0e8a
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Sat Oct 10 18:10:49 2015 -0400

    change bind-in to bind-then, remove bind

commit 1a95d291dfb4837d3db8917ba1fbf13d0af4ac38
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Sat Oct 10 18:13:54 2015 -0400

    organize structure

commit 36345fdad8ce4833610ea3c6fbf500c56af583fb
```

```
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Sat Oct 10 18:26:06 2015 -0400

    add .tar.gz target

commit 0582ba86d341cfd3b9e2fd0e9a8322322a1ba097
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Mon Oct 12 17:06:11 2015 -0400

    remove lots of ifs

commit eb2ecb7f87883811439bbac337b35d2f4c86fc71
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Mon Oct 12 17:06:37 2015 -0400

    remove lots of ifs

commit 79039e946ed81e7182d492992d6d5e70b04e09b4
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Mon Oct 12 17:07:26 2015 -0400

    lots of improvements

commit 1f5ad1a1899bca7e2accccc31e90a3c172d48f29
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Mon Oct 12 17:22:15 2015 -0400

    change seq delim to braces

commit 2423bd073a05efc2d316aed53c6e70007140018e
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Mon Oct 12 17:22:39 2015 -0400

    update eval rule

commit cd214c23d826223423a542d8cee2eb2465f833fd
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Mon Oct 12 23:09:48 2015 -0400

    add ifs again and fine-tuned error messages

commit 3fca92d6e65341559604b171c540dba962b05a35
Merge: 2423bd0 cd214c2
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Mon Oct 12 23:13:21 2015 -0400

    Merge pull request #1 from qwert42/rich-preds

    add ifs again and fine-tuned error messages

commit c305ffd451492aec8dbd04a4be069e3d16b62726
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Tue Oct 13 13:55:59 2015 -0400
```

```
    update makefile

commit 10e55e7578295da69a1d7fa914c85b9cacd4829b
Merge: 3fca92d c305ffd
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Tue Oct 13 13:56:27 2015 -0400

    Merge pull request #2 from qwert42/rich-preds

    update makefile

commit e74b5e1ac9a10b904bf094eb54c9f55721989676
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Wed Oct 14 20:11:03 2015 -0400

    stash

commit 7e9a8e97ed13db3cd545703cc4502eb2fe60af49
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Thu Oct 15 18:35:09 2015 -0400

    add bind also

commit 0bf11ece77103f2dcd4c0b0a1284b6ff03126e0c
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Fri Oct 16 13:46:20 2015 -0400

    add type declaration grammar

commit ea1c5b36d500664495fd7b0e5c751e522c19a72d
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Fri Oct 16 14:42:55 2015 -0400

    fix premature terminator

commit a8f41d13a1670659c028fecc8ba78b98cdabea1b
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Fri Oct 16 21:28:57 2015 -0400

    add altype literal grammar

commit e931e1541da05b287ff3983193274c0704291561
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Fri Oct 16 21:32:22 2015 -0400

    update adtsf grammar definition

commit 3bd462816f709a3c2a38823316bd389a8ca0dbf5
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Sat Oct 17 19:16:37 2015 -0400

    fix negative number scanning

commit 904b1e7c1ef966809702e4fa0ed37a92d423036b
```

```
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Sat Oct 17 19:17:04 2015 -0400

    add example for elements of programs

commit 4718ce6534ccc1098157268d4173901d805cc536
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Sat Oct 17 20:57:11 2015 -0400

    add more descriptions

commit 2e2d004476a4399e2569cce34a990de1328789e3
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Sun Oct 18 01:53:02 2015 -0400

    add something new

commit f42c3fa063286c4fe36eb17d24f178e9c6b6a425
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Sun Oct 18 12:14:23 2015 -0400

    fix escape sequence scanning

commit 4a7e0c63348616400d1b78c43b68dac0733e086a
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Sun Oct 18 22:56:31 2015 -0400

    fix typo

commit 6e87374d69a457e28622fa722c6c4a80793482e2
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Sun Oct 18 23:25:17 2015 -0400

    Update quicksort.w

commit 07d9e15f17591658ecb3819464f46834ca63f997
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Mon Oct 19 00:19:30 2015 -0400

    add comments for review

commit c1d21192764b8b53a01f8557862fa4681b4a18f3
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Mon Oct 19 00:21:06 2015 -0400

    Update quicksort.w

commit c7b6e801693b9a1a28f549403c3d2898bdf81521
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Mon Oct 19 00:51:13 2015 -0400

    Update gcd.w

commit cb9061eade02ffb6fa0adc0b278853ab7a873f4d
```

```
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Mon Oct 19 11:26:49 2015 -0400

    merge

commit 9e7c7271f7ee5f6671ae3dbcc70ffe0ac8d8ecb3
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Mon Oct 19 12:07:50 2015 -0400

    fix [Head]

commit a5ed176e26c7e754881ea8303d3cd45663c89ec8
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Mon Oct 19 12:09:37 2015 -0400

    remove push-list argument

commit 9dcf5e80b71e29f03b87baba9b9f27fbf39e270e
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Mon Oct 19 12:11:23 2015 -0400

    remove shared-seq push-stack

commit 05459f72ccc318d7b2abc855e4135b5af43bdcc6
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Tue Oct 20 17:23:22 2015 -0400

    change altype decl to postfix fashion

commit f8df1fdce3723b5c1997c70f2051af7a1118656b
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Tue Oct 20 17:23:30 2015 -0400

    add TODO

commit 9c611b38721436d09e35ae16910e0c2675f1f119
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Tue Oct 20 17:23:43 2015 -0400

    stash LRM

commit 081a6bc3f89ea783183f36e719d7481e0315e56b
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Thu Oct 22 00:13:18 2015 -0400

    add manual

commit 49f4a15948899a2cf4f006aba4589b77c423277b
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Thu Oct 22 16:52:17 2015 -0400

    hope this commit ends the paper work for a while

commit 4076cf8b28d8885f0049f5199e82519a68cd7760
```

```
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Thu Oct 22 21:29:41 2015 -0400

    add new docs

commit 35eb27e784c2b53d145a2e5492acdb66e7589298
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Thu Oct 22 21:32:12 2015 -0400

    add tail recursion and at symbol

commit 215cfb65051be7048acb7172c4eb4a4256e57b87
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Sat Oct 24 20:49:38 2015 -0400

    enhance number literals

commit b5713c0797a1fcd49ffe6c2861df0adc0a22b835
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Sun Oct 25 16:18:12 2015 -0400

    change to waf build system

commit b5d5d66d56bfba08c101231db0514499253d6b1b
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Sun Oct 25 17:40:33 2015 -0400

    restrict float token

commit 089a785c6745ca6ad082687d8e63d337457243ac
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Mon Oct 26 12:00:57 2015 -0400

    add readme

commit cba6266d372f576da854c35c98f0782655262214
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Mon Oct 26 12:01:11 2015 -0400

    change to python3

commit 706d95f32d050b69145ffb321cf7b80c28aba1c2
Merge: 4076cf8 cba6266
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Mon Oct 26 12:02:59 2015 -0400

    Merge pull request #3 from qwert42/tail-recursive

    Tail recursive

commit 3e16c66a222f878ce82beb7658ed90d4a97c1549
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Mon Nov 2 22:18:35 2015 -0500
```

```
    stash compiler prototype

commit a1f41c30d541332825c1e4653ba3f83633259bc7
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Mon Nov 2 22:19:03 2015 -0500

    stash manual

commit a566c4426da2fd7742df85785778b3142e196371
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Mon Nov 2 22:19:31 2015 -0500

    amend asm manual

commit 224355af65723d395ec5980e1e564ccaac704939
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Mon Nov 2 22:34:22 2015 -0500

    remove name_ref_key

commit d4d29c3bfce61b49326299cef1dbf1b0b5ab66d8
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Tue Nov 3 16:36:32 2015 -0500

    compiler 90%

commit f92a968bef03b61601b695b5bd971ef22cb4fa09
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Tue Nov 3 20:50:42 2015 -0500

    first somewhat working compiler version

commit 5eed46fa4cb3db014cf3670c7008cc286568b3df
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Tue Nov 3 21:34:59 2015 -0500

    add ufint and big int support

commit 5608397e1783fd9e70e6f49e5bd7e25f3d10720a
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Tue Nov 3 21:43:57 2015 -0500

    update manual

commit 504b7dd208f9545fb9b7c425cfbf25c4bdc36fcf
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Wed Nov 4 21:03:37 2015 -0500

    i just cant remember what i did

commit 61c023e02f7c76c7384d13b5ddfd6f6255e123b0
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Wed Nov 4 21:17:27 2015 -0500
```

```
    update readme

commit e5323f2f660f6f072a01e37778b50246c8ff2e66
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Thu Nov 5 20:24:57 2015 -0500

    a

commit 7a22edd382e9c1079f01da2aaffb8842f70c8360
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Sat Nov 7 22:32:47 2015 -0500

    fix certain bugs

commit e1e498d298ea09332214d4dea34884d7f1ee2294
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Sat Nov 7 23:04:48 2015 -0500

    resolve certain grammar issues

commit 55e0fc68f47c2b5ba521caac41b1670efeafb664
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Sun Nov 8 00:47:57 2015 -0500

    resolve some more issues

commit de01a09cd6e1f2743c88b50cb42424f6c6ef19cf
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Sun Nov 8 01:25:44 2015 -0500

    refine mechanisms about backquotes

commit 8d84f19b79930a783d448414c0dcb6f9d7664fc2
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Sun Nov 8 01:43:19 2015 -0500

    fix passing wrong inst ctx in g_seq

commit 56e99458c56a5d8122e65f8ea7aa6faaa4a007c3
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Sun Nov 8 12:37:26 2015 -0500

    fix some backquoted seq issues again

commit 3a77d8c0c9627712509584b201e124e2e6aa8526
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Sun Nov 8 12:40:03 2015 -0500

    Update cseg.ml

commit 988b7e6df18dc1334b36f36e40cb5be6878173a1
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Sun Nov 8 12:41:52 2015 -0500
```

```
    Update scoping.ml

commit db05d67ce3a053c6317ff39dad1dd0c8ac7795f4
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Sun Nov 8 12:59:38 2015 -0500

    change tasm to library

commit 9d7f5a2c3109f8e6899add7bd628e63133b2b327
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Sun Nov 8 13:06:48 2015 -0500

    change tasm to library

commit f37054c18ec1e1be5d579b8c02ef04923a461bbf
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Sun Nov 8 17:55:31 2015 -0500

    yeehaaaa

commit a7c35dd547dad29bf38c7343d141be408df558ca
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Sun Nov 8 18:23:55 2015 -0500

    sync instruction modifications

commit 8c65d6f6fa94611fdff8b8811b5ee3b44cbdfc70
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Sun Nov 8 20:05:53 2015 -0500

    add towel-mode

commit 96349b71f7646d374cac856ddf9d92c4676ca950
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Sun Nov 8 20:10:22 2015 -0500

    Update ccg.py

commit 1acd486b01fc013848e7c15c5c9a701b13ee93f1
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Sun Nov 8 20:11:23 2015 -0500

    Update TODO

commit 6bc36f0780b18a3ff2f9ada4a39d6f92a375a896
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Sun Nov 8 22:51:11 2015 -0500

    Update README.md

commit 47865a17230952ef3c1c4c3e99f76aa53ba08d59
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Sun Nov 8 23:28:40 2015 -0500
```

```
    Update README.md

commit 17152d5e68644c5278a903b3c2bccc573958c6f1
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Sun Nov 8 23:32:46 2015 -0500

    Update README.md

commit 0392e2fbd40d69b5230e34e6655aa238fb1dfdd4
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Mon Nov 9 00:13:59 2015 -0500

    Update README.md

commit d431dd33318ba718fac22fdd0c8311383be2d271
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Mon Nov 9 01:07:13 2015 -0500

    Update README.md

commit 3e5f696d503fb093ef8b8e1d1362af1a66a7e98f
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Mon Nov 9 14:27:27 2015 -0500

    reverse branch instructions

commit ec8d953081fccc88ed647d3c295b9f141772d7a0
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Mon Nov 9 17:42:16 2015 -0500

    change name indices to uint64

commit 78b1a46a0ceaa1f85a302861a520a41853c96c67
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Mon Nov 9 17:50:01 2015 -0500

    change jump pos to uint64

commit 70f94b2f1c4bdbe83e0f17f98776d425ad4cc832
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Tue Nov 10 14:49:51 2015 -0500

    add import and export phrases

commit e1b67b9ee1af813189232f40531411e1ab549b35
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Tue Nov 10 15:28:24 2015 -0500

    add import and export statements

commit 816a470e21d61333394fd544b9af4c5cda40fc62
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Tue Nov 10 15:30:54 2015 -0500
```

```
    stash vm changes

commit c9b80646072fc22ee6186648e0bef4deb45716d3
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Tue Nov 10 15:42:03 2015 -0500

    update gitignore

commit 4cf8afa4793b44371dfaa55f7dfb1e0d3c2b912d
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Wed Nov 11 12:35:16 2015 -0500

    add multiline comment

commit d37caf38e5a491084212563bb5d1685e083338e8
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Sat Nov 14 18:27:21 2015 -0500

    add prefect test suite

commit 111dda2c588477f02712f3b15511f79718aeddda
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Sat Nov 14 18:28:36 2015 -0500

    Update README.md

commit 16381dc0e8a54d680953ba135e6511c90bbe505b
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Sat Nov 14 18:29:25 2015 -0500

    Update wscript

commit a01a7f2d912ac871bd5516d51064c589a3d2969a
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Sun Nov 15 13:58:17 2015 -0500

    add .e files gen and import mechanism

commit 91c669dd9392cfaf17fa5f08497e0dae7e147162
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Sun Nov 15 23:11:07 2015 -0500

    hello world!

commit 01edd76fb943f31d4f63d974920da5c21d9cae2d
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Sun Nov 15 23:43:59 2015 -0500

    unquote scanned string

commit 2224d9951370714c0ef34f0cab982d9b1d18b2c6
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Mon Nov 16 00:18:57 2015 -0500
```

```
    amend prefect test suite

commit 819b6f63a34f0f360a713892433eea3fc623f0b5
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Mon Nov 16 00:26:37 2015 -0500

    amend prefect test suite

commit e3d94dc14eadd778a9c500866cfdb59d71d0423e
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Mon Nov 16 02:21:09 2015 -0500

    add branch instructions

commit 6a78a47f2d989611ef5c54baf418134c9842230b
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Mon Nov 16 15:19:18 2015 -0500

    (tail recursive) gcd working

commit bc2472f37f0eb661ffb9e012f24530e1ca97a5af
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Mon Nov 16 15:21:11 2015 -0500

    amend test description

commit 492a06a87b7e05316bd39d01e70d2df149fcb571
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Mon Nov 16 16:54:44 2015 -0500

    merge all the orts

commit 1ffcd2b5aaef9e68b2ad2af2ae209afbef119427
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Tue Nov 17 01:37:09 2015 -0500

    change to ccg.rb

commit e77b6a44673fb0bfb219135afdf3499c08d7ba64
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Tue Nov 17 12:38:04 2015 -0500

    add idle

commit cd3683c754b61939afd84d0c64bf7e7b67ba6e22
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Tue Nov 17 12:39:56 2015 -0500

    add pop

commit f9678e005a9db253cdbbf196be0b5e3c962b7d60
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Tue Nov 17 12:40:38 2015 -0500
```

```
    add idle 2

commit 20a65ee00228b9c244c14bf98475121a046a4181
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Tue Nov 17 12:41:13 2015 -0500

    fix branching instruction label args

commit 132a1dd873418be434f00887d87f175d4ab47e19
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Tue Nov 17 12:42:07 2015 -0500

    dont remember

commit 79f36dfbfebe9dcf4806845d0fcc082c9e215c8e
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Tue Nov 17 12:42:23 2015 -0500

    update readme

commit 3b37049e0cf3992fb89130df6a70f9024700cf4f
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Tue Nov 17 12:42:34 2015 -0500

    add sqrt example

commit 360cd649f1bbb604b500df51d2cbbb83411c065b
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Tue Nov 17 12:49:58 2015 -0500

    change std naming standard

commit 0087e38f969c49d3562cebf463ed1865f5be3c05
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Tue Nov 17 15:31:19 2015 -0500

    prepare for generalized arithmetics

commit c33f631c999cbe8749996323a8669c092eec8dca
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Tue Nov 17 15:43:50 2015 -0500

    fix import mechanism

commit bee96ae57f55adec4a03dc86548d206db1fb8bc4
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Tue Nov 17 20:37:32 2015 -0500

    add std_gen

commit 1bbb971cdc5d160b00a948ae2cca5e5554b36633
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Sat Nov 21 19:17:16 2015 -0500
```

```
    add function backquote

commit 98c087aea73d1b95c49aefc3a32eb93763f84645
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Sat Nov 21 21:51:50 2015 -0500

    add push-lit

commit a9e0c4b9c33b26e2d1b81ba1b0599476429531d8
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Sat Nov 21 21:53:13 2015 -0500

    add partial fun-arg

commit 138f0cb5a1f80217e993d75cf5706e897023c300
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Sat Nov 21 21:53:37 2015 -0500

    add stacked list/tuple creation

commit 57cd0428ff4646e34e22df1845e039165c8dec9c
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Mon Nov 23 11:35:24 2015 -0500

    add closure set to fun_t, remove them from funpool

commit 1de084738b6c5ad2e2482f7c49ae72a1f08bf818
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Mon Nov 23 11:35:58 2015 -0500

    finish std-gen script

commit 8131e4007ddc62a335af81a1810059336cfa2b3f
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Mon Nov 23 11:36:31 2015 -0500

    recover to *-tuple and *-list insts

commit a78410b0bde4ade7854b3f06e813985a5692bc06
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Mon Nov 23 11:36:53 2015 -0500

    fix ref number to BQFun

commit f7cefd74055547cda9abfefeef7de5c18eeab4af
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Mon Nov 23 11:37:25 2015 -0500

    add VAtom to tasm_ast.mli

commit 383bdc857a4133f682efb32745ffced29bd969d1
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Mon Nov 23 12:19:23 2015 -0500
```

```
    fix atom stringifying

commit 4e0c9f4760998268ce8d6918af1b33c84d3b4218
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Mon Nov 23 12:19:57 2015 -0500

    fix true and false atom_repr

commit cc117db2422032646dd042aad675f34c5aafc643
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Mon Nov 23 12:20:08 2015 -0500

    fix warnings

commit b215176231833466c05d183eba1b644d608e3042
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Mon Nov 23 12:52:04 2015 -0500

    add ift-f and import tests

commit deadfc7d3ea8fd3d36c75128268beddfc614c1e5
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Mon Nov 23 13:14:17 2015 -0500

    add phony (stack bottom indicator)

commit 704eb448875515ef6fc6c7cb2bf571a8cc8a8f3f
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Mon Nov 23 15:06:16 2015 -0500

    add phony

commit afdb02c69f629c80c3add57589fcd7fde1c6f32b
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Wed Nov 25 14:51:14 2015 -0500

    stash changes

commit 5b1947dea3e5e6501d14631a622624e31291972c
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Thu Nov 26 02:26:30 2015 -0500

    get rid of some warnings

commit dd83c647ede6424a27e364967174be0746f69bf8
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Thu Nov 26 02:27:24 2015 -0500

    refactor

commit 01e1ab3a73a19385065a128954e4b955cf9a2b40
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Thu Nov 26 02:28:47 2015 -0500
```

```
    cope with new function grammar

commit 94c2aeb06d50f904710c1be75a05784ed3bfc12a
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Sun Nov 29 00:03:34 2015 -0500

    remove hjne

commit b8db2dd469e0a785e3417fec2b146bc8be8ed2cc
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Sun Nov 29 00:04:55 2015 -0500

    refactor vm

commit f7dbbbffb6aac8dc492dddd33994c605d0ac7ea5
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Sun Nov 29 00:05:49 2015 -0500

    add new highlighting

commit 305c0b2bef2317ca06a92c8351ae0b22b470b109
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Sun Nov 29 16:05:09 2015 -0500

    resolve regression

commit 39cb55cd44d8995ffeef795247e624b99399c3f5
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Sun Nov 29 16:06:04 2015 -0500

    add !rev built-in

commit 354ef3a31340eb1f66b3417665a34a5fa15f46f3
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Sun Nov 29 16:35:08 2015 -0500

    refine seq code generation

commit e347db7536184ffed753ff91769377ca893c0592
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Sun Nov 29 16:35:40 2015 -0500

    redef reverse and pack arity

commit 905648856ca365c17c3609432686aba90c0c9e14
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Sun Nov 29 18:21:22 2015 -0500

    remove equ instructions

commit 19d4ca85e302021a926ca275a918727eb996ce7c
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Sun Nov 29 18:21:54 2015 -0500
```

```
    add abs module id to avoid import collision

commit cfc2e9b603514232ff5374b7b68eee1fbcd09235
Merge: 19d4ca8 704eb44
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Sun Nov 29 18:25:40 2015 -0500

    merge to master

commit 97f57a4385309d31608445079afe0e6c82d1a785
Merge: 704eb44 cfc2e9b
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Sun Nov 29 18:26:08 2015 -0500

    Merge branch 'ortless'

commit 8a90269f19e019129abe57ae095bd3f32fa7cac3
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Sun Nov 29 18:41:14 2015 -0500

    Update README.md

commit 3155431d8637e3a40a0a670fb0fe031a62435799
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Sun Nov 29 21:46:42 2015 -0500

    fix warning

commit c13b79ce59bbb822c073a113c7ce562ea3674399
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Sun Nov 29 21:47:30 2015 -0500

    remove about 20 instructions

commit 8a4c3b944fdc16e5f20c4fae86744ae28686ce65
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Sun Nov 29 22:45:14 2015 -0500

    add push-list etc.

commit bf1d398fb3026e497a344051e332d5226c421868
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Sun Nov 29 22:45:27 2015 -0500

    add partial fun test

commit 6f362d06f4101a4e777eff42f83545a58335a984
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Mon Nov 30 13:04:38 2015 -0500

    add invoke inst

commit bff1a4601b419bead8c7a4b4d531dc48c5605ded
Author: Zihang Chen <chsc4698@gmail.com>
```

```
Date:   Mon Nov 30 13:05:09 2015 -0500

    remove OVLNil and OVTNil

commit 051116e3a287eaaf833a94455a45f250e08c8e9b
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Mon Nov 30 20:23:41 2015 -0500

    rename make-fun and push-fun

commit b80d4cc434f8562a9826d9c8f30234c4d6651369
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Mon Nov 30 20:44:18 2015 -0500

    fix potential partial fun arg issue

commit 21a6c5c79315e80d09bc7f351e239db6c34aec3e
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Mon Nov 30 20:54:21 2015 -0500

    rm unused file

commit 3f1ff9e3a42b2177a00e8500e19b8b90cd9298a2
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Mon Nov 30 21:36:07 2015 -0500

    remove period terminator (really no point in this)

commit b2364f1e4cd6fc5a71492f97c0d3b693578c6095
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Mon Nov 30 21:36:59 2015 -0500

    add list operations

commit 2fa2ef3373e982f8220281a7f660d917a07b2be1
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Tue Dec 1 00:08:27 2015 -0500

    add type sig to std lib

commit 8942b9f92cdc5b3673cc817ba2a5c4051f7e26fe
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Tue Dec 1 14:21:19 2015 -0500

    add cons

commit c145ac9fbc78400a337e8d16a120e1b549651bae
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Tue Dec 1 14:22:38 2015 -0500

    hide std lib funs

commit 6b500686fa76119eacbaaf189b59e3e85f0d3f36
Author: Zihang Chen <chsc4698@gmail.com>
```

```
Date:   Tue Dec 1 14:23:01 2015 -0500

    handle compile failure and exec failure

commit bb6b840307475a9b774bc715bcba4fc008565f56
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Tue Dec 1 15:57:01 2015 -0500

    fix bind push-name

commit 19e96d2643bb64fec3563e498259a2dec2aa7a23
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Tue Dec 1 15:57:47 2015 -0500

    get rid of dot at the end of .e files

commit f8eee5e5109b4f29f0d48f54eedb71de082a9540
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Tue Dec 1 15:59:26 2015 -0500

    add new std lib, but broke imp import

commit 781602e1f1ce374f30e357fc9f85f20265f2e7e7
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Tue Dec 1 17:23:17 2015 -0500

    fix imp import by counting name ids

commit 1ca7baef5176b94e2cfc4932a1d37474e2ed6d78
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Wed Dec 2 14:26:34 2015 -0500

    fix relative importing and name resolving

commit fe762e4a3c3d928b0d5a03fabdab1e6298d1519c
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Wed Dec 2 14:28:13 2015 -0500

    change to rel import

commit 529325a262fee81d559887bcc52fb0976354c779
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Wed Dec 2 14:28:31 2015 -0500

    add mod and equ

commit c834997d2e65d0e49882ce725a302e8d9397c35a
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Wed Dec 2 14:29:14 2015 -0500

    add functional library fun and test

commit b714c0dccb8f86e0ee80cdb1325b6021dd0f7709
Author: Zihang Chen <chsc4698@gmail.com>
```

```
Date:   Wed Dec 2 14:56:19 2015 -0500

    refine EQU, add :filter

commit 4699eaa9ddbbec794aa421732baa23cff5e9ed36
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Wed Dec 2 15:13:24 2015 -0500

    add travis

commit 2916cc76f6437b0bdace50f45a41e746d9081ef5
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Wed Dec 2 15:18:30 2015 -0500

    set sudo: true

commit 7c4c883c266059c559a1305d0e00a9b5f0ed9196
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Wed Dec 2 15:21:59 2015 -0500

    change to source install opam

commit dc3f120def68622d9b3b77f0665ad20f6e1381a0
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Wed Dec 2 15:28:34 2015 -0500

    change to source install opam

commit 389ee7fd9794d74600235e9867c9d10bec4f52d9
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Wed Dec 2 15:31:21 2015 -0500

    .

commit 05c1de2f120cb7a07f5f74cb446737cd745b196a
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Wed Dec 2 15:33:42 2015 -0500

    fix typo

commit 4328355ae6dcb6cb5f3ee6eb10d54f2d008d4b98
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Wed Dec 2 15:39:59 2015 -0500

    fix ruby install

commit d91d288efdc6c9b279435f4f20ce3d3ad8112f2b
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Wed Dec 2 19:50:12 2015 -0500

    fix seq label issues

commit 548173944a670593fe3058e71e5e746e04a18a70
Author: Zihang Chen <chsc4698@gmail.com>
```

```
Date:   Wed Dec 2 19:50:48 2015 -0500

    add /flip and /foldr

commit b282f4f673f8b3768cd3d4f06fb1ec997d0ebbc7
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Wed Dec 2 19:57:21 2015 -0500

    fix recursive fun args

commit 873b98a6ee548c645040d83627743951dd182819
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Wed Dec 2 20:05:03 2015 -0500

    update comment

commit 516174a90141904bb3b5b8860b0623d5fd9fa1f9
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Wed Dec 2 20:24:40 2015 -0500

    add lambda token

commit 5ca39838852d2044bf294e440e1282257290a678
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Wed Dec 2 21:34:06 2015 -0500

    fix branching insts

commit cec55ebbd89eb147eef9f74bddd1b61d6d4bcf1f
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Wed Dec 2 21:36:48 2015 -0500

    fix phony mechanism

commit 13110e8f8f0d1563cfc3eda30e9da82613fecf9f
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Wed Dec 2 22:16:17 2015 -0500

    add aliases to list functions

commit 901df6d3ac29e9299b0aa207900501de25ffbfa5
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Wed Dec 2 23:23:08 2015 -0500

    .

commit d3a6dcaae2018e114d1e83fdb059fe3c040f8d25
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Wed Dec 2 23:32:07 2015 -0500

    add .gitattributes

commit 3ebcb3cda7a7f217edc493c22e30b26cc6417862
Author: Zihang Chen <chsc4698@gmail.com>
```

```
Date:   Thu Dec 3 14:41:12 2015 -0500

    fix rel import name resolving

commit 652e9c30583ce174d703de65dd1517942641c9b8
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Thu Dec 3 14:45:00 2015 -0500

    support test suppression by prefixing __

commit 85242c0892e452c551b66aa69d86d9bc2163fc86
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Thu Dec 3 14:46:35 2015 -0500

    add built-in and some stack insts

commit bfe25336ff1e4cdfab02c03009bd2332d1c6769d
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Thu Dec 3 19:26:25 2015 -0500

    fix closure binding

commit a6c30f7ff740c61bafb78ee419815ef6f82e394e
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Thu Dec 3 19:55:10 2015 -0500

    implement ext module mechanism

commit 5646e45ae7fb541b07376e7a19b63bb8a6234c1e
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Thu Dec 3 20:04:16 2015 -0500

    add new gitignore entries

commit 156b5c6a0c0faf619a6d74b6dfbbfdda84d94ae9
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Thu Dec 3 21:39:34 2015 -0500

    add something new

commit 00d86e6fa1a8f7883c750cc04cbfb3f5020cd159
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Thu Dec 3 21:45:41 2015 -0500

    Update README.md

commit b95f09835365db65c1be98bbe3ba63af1b0e9207
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Thu Dec 3 23:51:32 2015 -0500

    fix je and hje, remove built-in

commit db7d9c87ec4d7b96df096c27f1c53e0d334b5d28
Author: Zihang Chen <chsc4698@gmail.com>
```

```
Date:   Fri Dec 4 00:33:13 2015 -0500

    amend towel asm defs

commit b444ef28d99c1da86083941a12869178c609cd97
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Fri Dec 4 00:55:19 2015 -0500

    add ext howto

commit ce8a6feb02f26f3e4164e29ffb8e2427f96407be
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Fri Dec 4 00:56:25 2015 -0500

    remove sh files

commit a09fd42d932eae637b2b6026379ea47b17f84d8c
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Fri Dec 4 01:00:16 2015 -0500

    refine prefect

commit b8cf11a7352557048dae5e8dc8bd02c4c8921204
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Fri Dec 4 01:02:04 2015 -0500

    update readme

commit 41a86a44f4b1426495a440342538c6eb15f7aa2f
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Fri Dec 4 03:23:54 2015 -0500

    update towel asm manual

commit 87293913668a22375078420e2f1c9078e2a5cf46
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Fri Dec 4 03:24:23 2015 -0500

    hello world tk!

commit f735768503b8f6b6ca67a0702b1fbd23fc0bc244
Merge: 2fa2ef3 8729391
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Fri Dec 4 03:24:43 2015 -0500

    Merge pull request #6 from qwert42/dynatowel

    Dynatowel!

commit 507bdd554e4f80e10d7471780be633c934b893be
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Fri Dec 4 03:27:56 2015 -0500

    Update README.md
```

```
commit 7f9244d7c81efc7b729fdf08d31a72cdbfca9833
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Fri Dec 4 03:28:16 2015 -0500

    Update README.md

commit 163a5f2835bac4ec9c514b04ef6c87052e36e9d5
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Fri Dec 4 03:29:09 2015 -0500

    update towel asm manual

commit 1ad889deaa22b89ccf5f1e98f9f9368c0f090043
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Fri Dec 4 12:40:49 2015 -0500

    Update std_gen.rb

commit af387a8e6b80ecee89a285a97846f4f6319f72ce
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Fri Dec 4 13:33:19 2015 -0500

    change ~pred to ?pred

commit 38586b2f52f0a682eaad359ed3a1b9a4b577b499
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Fri Dec 4 13:34:05 2015 -0500

    update TODO

commit ac954d940d2d7e414ad8f86384194f773d854e2c
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Fri Dec 4 22:57:54 2015 -0500

    make some amend

commit da1de6834be64c68ea7f7d003dfa16a1a5a43065
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Sun Dec 6 01:09:12 2015 -0500

    add bytecode representation

commit d4f37a6da1f1e1bd99700d100cbdf6fa10071d1e
Merge: 38586b2 da1de68
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Sun Dec 6 01:12:16 2015 -0500

    Merge pull request #7 from qwert42/bytetowel

    Bytetowel

commit 7009ae603c7be6d59af1a7090951f42f4d1007cc
Author: Zihang Chen <chsc4698@gmail.com>
```

```
Date:   Sun Dec 6 01:14:44 2015 -0500

    lose unnecessary dep

commit 38f75bf930408e8c9ff50bc7508365a45ef8bbfb
Merge: d4f37a6 7009ae6
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Sun Dec 6 01:15:18 2015 -0500

    Merge pull request #8 from qwert42/bytetowel

    lose unnecessary dep

commit 0bfc5a5172ad7d638ed412afb463621487fb98ce
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Sun Dec 6 10:31:48 2015 -0500

    remove idle and phony from grammar, add them to .w

commit 5305ba6d176d83e90b30e314ea6cdb3d4b741902
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Sun Dec 6 13:44:32 2015 -0500

    fix tk bytecode parsing error

commit 50777b2378654ea411b4c27235d4d01a01af62f2
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Sun Dec 6 17:38:50 2015 -0500

    refactor name resolving

commit 14112bb342ce799fc41ed558a3fb8db63cebaa2a
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Mon Dec 7 00:58:28 2015 -0500

    add install instruction, quicksort still fails

commit f66076f650d698ab796ed3b60272c87c14c24481
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Mon Dec 7 00:58:57 2015 -0500

    opt std stack usage

commit fb4c2794d20845cb825e331058b5a8dfd0ce3bbd
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Mon Dec 7 00:59:08 2015 -0500

    add *** debug instruction

commit 460551dc02d3c88aa45879ffd1e3f5aeddd6c314
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Mon Dec 7 00:59:50 2015 -0500

    add tests; quicksort.t still fails
```

```
commit 9f2388ba1fe117347fe6482ca5a517b24e55fc12
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Mon Dec 7 01:00:47 2015 -0500

    add one more test

commit adb1d3c7520d7ef4fb066ef9cb92d2493837986a
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Mon Dec 7 12:49:49 2015 -0500

    fix scope overflow

commit ec64e870e210f65d5cdf0e151648bc8e3630a503
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Mon Dec 7 13:14:32 2015 -0500

    add more wrappers for .w

commit 9092e09ff9b7b8ed2fc75b150a736ae9df45f1b8
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Mon Dec 7 23:16:04 2015 -0500

    add type reflection, opt name resolve

commit 1fc75f2ac775b26c260e2ff9f84308ddd59f48cb
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Tue Dec 8 01:15:09 2015 -0500

    remove all the compiler-side type decls

commit b19997c08908aabde7940a5ddc0d68657539b505
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Tue Dec 8 19:02:41 2015 -0500

    change list inflation mechanism

commit d4df4251fc4f2c2b7a12727ca389795b7a96180f
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Tue Dec 8 19:13:52 2015 -0500

    be conservative about put_val

commit fa8caf0c0b04e27cae92c3513f2b67d10291d67c
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Tue Dec 8 23:56:30 2015 -0500

    add tuple-at

commit 1eaf57d5e94b5e7924258b267fc820b4dc264d81
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Wed Dec 9 15:47:10 2015 -0500

    revise manual
```

```
commit dbbe42bcfd500f2f199cc6154d13c99670a5e09d
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Thu Dec 10 02:17:23 2015 -0500

    modify import instruction format

commit b2910cbb2828487174e59efba7771c9e72f53544
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Thu Dec 10 02:17:39 2015 -0500

    add report

commit 8c8234d4d286ff65c7294121944c28c35e4b7555
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Thu Dec 10 11:38:40 2015 -0500

    change copy to ref in closure

commit 6999be5b9484309ff2a608483976a4079a965d57
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Thu Dec 10 15:16:26 2015 -0500

    add more tests and texts

commit e4afa3ac4267fef480796f0bfbca98a35748cf9a
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Thu Dec 10 21:01:35 2015 -0500

    stash

commit ab90afdcd77be7b0194904590e8f051d0590814e
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Fri Dec 11 00:02:59 2015 -0500

    stash report

commit 7048cbf36da1780e851caedcc9c508718222f3ce
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Fri Dec 11 02:20:35 2015 -0500

    stash

commit 860e7a24befa3b32fa2169f6ef943f5d0307f840
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Fri Dec 11 02:33:59 2015 -0500

    remove unused code

commit 213e8d4e543146d524007d6cc5724b270b52d123
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Fri Dec 11 02:35:57 2015 -0500

    remove unused file
```

```
commit d03b3b0a48eb74884bdbeb78c53e97159ff011bc
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Fri Dec 11 02:40:17 2015 -0500

    update report.tex

commit d67eec4642356d4a7dbe116695b8ea51417d9d24
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Fri Dec 11 19:12:11 2015 -0500

    remove outdated structs

commit 0bc1a99adea9ddac08dac85f23404b342a6a979a
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Fri Dec 11 19:12:57 2015 -0500

    add arithmetic instructions

commit c7bce586869557537b0bbe2d133133f6a72afbff
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Fri Dec 11 19:13:21 2015 -0500

    stash report

commit ccdf962de72dc951c9f0cd06b457cccd24928ed6
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Fri Dec 11 23:48:37 2015 -0500

    handle return value and enumerable literals

commit 0e6902d0ba4acc93e4dc3de210d43ed70a5c5aed
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Fri Dec 11 23:49:12 2015 -0500

    stash report source gen script

commit ae9e1f0d88b17ef60e56dee1288af34472389709
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Fri Dec 11 23:49:57 2015 -0500

    amend comments

commit 041dc56de338bb9e2c75d3c17b364d9463e37490
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Fri Dec 11 23:50:28 2015 -0500

    fix float regex

commit fbccf86feac0b11e37ece64b1e424760fe6ac474
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Fri Dec 11 23:50:55 2015 -0500

    rename tuple indexing functions
```

```
commit 546db190ef1749a841b9376bf38963227cd3e0eb
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Fri Dec 11 23:51:18 2015 -0500

    fix shift instructions

commit a4c8fe18856ba1a8002e690d0b8a4d738f951a01
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Sat Dec 12 01:13:02 2015 -0500

    add sweep for tail recursive stacks

commit 8744518135c8505ad7c5ff539c96d905ca74763c
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Sat Dec 12 01:13:18 2015 -0500

    fix is_tail_recursive_flag

commit 2339d6183305bf95d6b75657f3cd063d3be002dd
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Sat Dec 12 01:13:35 2015 -0500

    add zip.t

commit 70589608b786f608e89d164b93b436f38549d81a
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Sat Dec 12 02:05:27 2015 -0500

    remove !!pop as much as possible

commit 57ac0466b72ada86cc0e2893d887828f6fc2dbb2
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Sat Dec 12 02:06:41 2015 -0500

    stash report

commit d53392183cdc82cb283548ea795ecbde01cfeab5
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Sat Dec 12 13:50:07 2015 -0500

    fix bind in find_closure

commit 71e816156d77f54fcd91d72a571c3ec370c19a30
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Sat Dec 12 13:50:18 2015 -0500

    stash report

commit 87f3fb75f632c2dc1793c4cc127f65087c31b141
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Sun Dec 13 19:36:43 2015 -0500

    stash
```

```
commit ec400948288a229e955ac2300cc5ad1df2830dad
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Mon Dec 14 19:21:53 2015 -0500

    stash report

commit 78d3d89047aa45ac84242a4dbcbe2e30194e9992
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Tue Dec 15 12:40:59 2015 -0500

    add pack all (-1)

commit 1d32cff53a3bc4600deeb0973555f8db22bdb771
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Tue Dec 15 12:51:27 2015 -0500

    add string hd tl empty cons

commit 2a5c398000cbf906056b8b402d053cc202e9bf61
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Tue Dec 15 13:08:04 2015 -0500

    change tuple pack to list pack

commit a055564b6e56e9944ab2102615e50702014134b4
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Wed Dec 16 18:24:41 2015 -0500

    update comment

commit 9e8f92eb053b52525f35f22fce4134772f4b74e4
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Wed Dec 16 18:25:02 2015 -0500

    fix backquoted shared sequence closure

commit 81c7c2f709e0895e91e3e976bf3b20fa9588f684
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Wed Dec 16 18:25:18 2015 -0500

    add variadic functions

commit 8e04893a82ec91389924d92a740f34da191fc74d
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Wed Dec 16 18:25:33 2015 -0500

    add wrappers for push-*nil and end-*

commit 1e90a5715e26e3c64219df169ff903a932e39eab
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Wed Dec 16 18:26:16 2015 -0500

    add vargs test
```

```
commit 59a45742ffb85d09ced26b293b7cc734b17e6fab
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Wed Dec 16 18:26:31 2015 -0500

    stash report

commit 1a462945b90aba6a79559fe35e043b755f2e6428
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Wed Dec 16 22:13:11 2015 -0500

    move to bsd license from gpl

commit e83e8a0598423643c0c8fa353bb7a1f463e9a525
Author: Zihang Chen <chsc4698@gmail.com>
Date:   Wed Dec 16 22:13:29 2015 -0500

    stash report
```

Appendix B

Full Code Listing

B.1 License Information

This project is licensed under the terms of BSD 3-Clause License.

```
Copyright (c) 2015, Zihang Chen (qwert42, chsc4698@gmail.com)
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice,
this list of conditions and the following disclaimer in the documentation
and/or other materials provided with the distribution.

3. Neither the name of the copyright holder nor the names of its contributors
may be used to endorse or promote products derived from this software without
specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE
LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
POSSIBILITY OF SUCH DAMAGE.
```

Hint

Source code is also available online at `https://github.com/qwert42/Towel`.

B.2 `compiler/assemble.ml`

Code Listing

```
(* compiler/assemble.ml -- Author: Zihang Chen (zc2324) *)
open Tasm_ast;;
open Stdint;;

let lbl:(string, uint64) Hashtbl.t = Hashtbl.create ~random:true 512;;

let replace_label =
  function (* I could have generate this with script. But,... nah... *)
  | JUMP(ArgLabel(Label(s)))
    -> JUMP(ArgLit(VUFixedInt(Hashtbl.find lbl s)))

  | JGEZ(ArgLabel(Label(s)))
    -> JGEZ(ArgLit(VUFixedInt(Hashtbl.find lbl s)))
  | HJGEZ(ArgLabel(Label(s)))
    -> HJGEZ(ArgLit(VUFixedInt(Hashtbl.find lbl s)))
  | JGZ(ArgLabel(Label(s)))
    -> JGZ(ArgLit(VUFixedInt(Hashtbl.find lbl s)))
  | HJGZ(ArgLabel(Label(s)))
    -> HJGZ(ArgLit(VUFixedInt(Hashtbl.find lbl s)))

  | JLEZ(ArgLabel(Label(s)))
    -> JLEZ(ArgLit(VUFixedInt(Hashtbl.find lbl s)))
  | HJLEZ(ArgLabel(Label(s)))
    -> HJLEZ(ArgLit(VUFixedInt(Hashtbl.find lbl s)))
  | JLZ(ArgLabel(Label(s)))
    -> JLZ(ArgLit(VUFixedInt(Hashtbl.find lbl s)))
  | HJLZ(ArgLabel(Label(s)))
    -> HJLZ(ArgLit(VUFixedInt(Hashtbl.find lbl s)))

  | JEZ(ArgLabel(Label(s)))
    -> JEZ(ArgLit(VUFixedInt(Hashtbl.find lbl s)))
  | HJEZ(ArgLabel(Label(s)))
    -> HJEZ(ArgLit(VUFixedInt(Hashtbl.find lbl s)))
  | JNEZ(ArgLabel(Label(s)))
    -> JNEZ(ArgLit(VUFixedInt(Hashtbl.find lbl s)))
  | HJNEZ(ArgLabel(Label(s)))
    -> HJNEZ(ArgLit(VUFixedInt(Hashtbl.find lbl s)))

  | JT(ArgLabel(Label(s)))
    -> JT(ArgLit(VUFixedInt(Hashtbl.find lbl s)))
  | HJT(ArgLabel(Label(s)))
    -> HJT(ArgLit(VUFixedInt(Hashtbl.find lbl s)))
  | JF(ArgLabel(Label(s)))
    -> JF(ArgLit(VUFixedInt(Hashtbl.find lbl s)))
  | HJF(ArgLabel(Label(s)))
    -> HJF(ArgLit(VUFixedInt(Hashtbl.find lbl s)))

  | JE(ArgLabel(Label(s)))
    -> JE(ArgLit(VUFixedInt(Hashtbl.find lbl s)))
  | JNE(ArgLabel(Label(s)))
    -> JNE(ArgLit(VUFixedInt(Hashtbl.find lbl s)))
```

```
52    | HJE(ArgLabel(Label(s)))
53      -> HJE(ArgLit(VUFixedInt(Hashtbl.find lbl s)))
54    | HJNE(ArgLabel(Label(s)))
55      -> HJNE(ArgLit(VUFixedInt(Hashtbl.find lbl s)))
56
57
58    | PUSH_FUN(ArgLabel(Label(s)))
59      -> PUSH_FUN(ArgLit(VUFixedInt(Hashtbl.find lbl s)))
60    | CALL(ArgLabel(Label(s)))
61      -> CALL(ArgLit(VUFixedInt(Hashtbl.find lbl s)))
62
63    | _ as x -> x;;
64
65  let rec _inflate cnt =
66    let to_strs = List.map (fun x -> match x with Label(l) -> l)
67    in let _reg labels rest =
68         let none_of_the_labels_are_registered =
69           List.fold_left (&&) true
70           @@ List.map (fun x -> not (Hashtbl.mem lbl x)) @@ to_strs labels
71         in if none_of_the_labels_are_registered
72         then let () =
73                List.iter (fun x -> Hashtbl.add lbl x cnt) @@ to_strs labels
74           in _inflate (Uint64.succ cnt) rest
75         else failwith "Redefining labels, exiting."
76
77    in function
78        [] -> ()
79      | CLine(labels, _)::rest ->
80        _reg labels rest
81      | Line(labels, _)::rest ->
82        _reg labels rest;;
83
84  let assemble asm =
85    match asm with
86      Asm(ls) ->
87      let () = _inflate Uint64.zero ls
88      in Asm(List.map
89               (fun x -> match x with
90                    Line(_, inst) -> Line([], replace_label inst)
91                  | CLine(_, Some(inst)) -> Line([], replace_label inst)
92                  | CLine(_, None) -> CLine([], None))
93               ls)
```

B.3 compiler/ast.mli

Code Listing

```
1  (* compiler/ast.mli -- Author: Zihang Chen (zc2324) *)
2  open Stdint
3
4  type atom = {
5    atom_name: string;
```

```
  atom_repr: int
}

and pname = {
  name_repr: string;
}

and name =
    NRegular of pname list
  | NTailCall of pname list

type pvalue = {value_content: pvalue_content}
and pvalue_content =
    VAtom of atom
  | VFixedInt of int64
  | VInt of Big_int.big_int
  | VUFixedInt of uint64
  | VFloat of float
  | VList of word list
  | VString of string
  | VTuple of word list

and backquote =
    BQValue of pvalue
  | BQName of name
  | BQSeq of sequence
  | BQBackquote of backquote

and import =
    ExplicitImport of string list
  | ImplicitImport of string list

and word =
    WLiteral of pvalue
  | WName of name
  | WBackquote of backquote
  | WSequence of sequence
  | WControl of control_sequence
  | WFunction of function_sform
  | WImport of import
  | WExport of pname list
  | WBind of bind_sform

and sequence =
    SharedSequence of word list
  | Sequence of word list

and if_body = IfBody of word * word
and if_sform =
    IfGEZ of if_body
  | IfGZ of if_body
  | IfLEZ of if_body
  | IfLZ of if_body
```

```
    | IfEmpty of if_body
    | IfNonEmpty of if_body
    | IfEZ of if_body
    | IfNEZ of if_body
    | IfT of if_body
    | IfF of if_body

and control_sequence =
    CtrlSeqIfForm of if_sform

and arg_def =
    ArgDef of pname

and function_sform =
    Function of arg_def list * word
  | BQFunction of arg_def list * word

and bind_body = BindBody of pname * word

and bind_sform = BindThen of bind_body list * word

and terminator = EOF

type sentence = Sentence of word list;;
```

B.4 compiler/common.ml

Code Listing

```
(* compiler/common.ml -- Author: Zihang Chen (zc2324) *)
open Batteries
open Ast
open Stdint

(* ======================================
     Counter
   ====================================== *)
let counter = fun () ->
  let cnt = Array.of_list [Uint64.of_int 0]
  in fun () -> cnt.(0) <- Uint64.succ cnt.(0); cnt.(0);;
(* Losing one -1 for a whole bunch of 2**63. *)

let tu64 x = Printf.sprintf "%su" @@ Uint64.to_string x;;

(* ======================================
     AST stringifiers
   ====================================== *)
module P = Printf;;

let rec atom_stringify a = P.sprintf "(atom %s %d)" a.atom_name a.atom_repr
and pname_stringify pn = P.sprintf "(name %s)" pn.name_repr
and name_stringify name = let a = match name with
```

```
      NRegular(ns) -> ns
    | NTailCall(ns) -> ns
  in String.concat " of " (List.map pname_stringify a)
and int_stringify a = P.sprintf "(int-lit %s)" @@ Int64.to_string a
and float_stringify a = P.sprintf "(float-lit %f)" a
and string_stringify a = P.sprintf "(str-lit %s)" a
and tuple_stringify a = P.sprintf "(tuple-lit %s)"
    (String.concat ", " (List.map word_stringify a))
and lit_stringify = function
  VAtom(a) -> atom_stringify a
| VFixedInt(i) -> int_stringify i
| VFloat(f) -> float_stringify f
| VList(l) -> list_stringify l
| VString(ss) -> string_stringify ss
| VTuple(ws) -> tuple_stringify ws
| _ -> "i'm just too lazy"

and seq_stringify seq =
  match seq with
    Sequence(ws) -> String.concat ", " (List.map word_stringify ws)
  | SharedSequence(ws) -> String.concat "-, " (List.map word_stringify ws)

and backquote_stringify a = P.sprintf "(bq-lit %s)"
    (match a with
       BQValue(pv) -> lit_stringify pv.value_content
     | BQName(n) -> name_stringify n
     | BQSeq(seq) -> seq_stringify seq
     | BQBackquote(bq) -> backquote_stringify bq)

and cs_stringify cs =
  let if_stringify s body =
    match body with
      IfBody(w1, w2) -> P.sprintf "%s { %s; %s }" s
                          (word_stringify w1)
                          (word_stringify w2)
  in match cs with
    CtrlSeqIfForm(i) ->
    (match i with
       IfGEZ(ib) -> if_stringify "gez" ib
     | IfGZ(ib) -> if_stringify "gz" ib
     | IfLEZ(ib) -> if_stringify "lez" ib
     | IfLZ(ib) -> if_stringify "lz" ib
     | IfEmpty(ib) -> if_stringify "empty" ib
     | IfNonEmpty(ib) -> if_stringify "non-empty" ib
     | IfEZ(ib) -> if_stringify "ez" ib
     | IfNEZ(ib) -> if_stringify "nez" ib
     | IfT(ib) -> if_stringify "t" ib
     | IfF(ib) -> if_stringify "f" ib)

and arg_def_stringify d =
    match d with
      ArgDef(n) -> pname_stringify n

```

```
and bind_body_stringify = function
    BindBody(n, w) ->
    P.sprintf "%s = %s" (pname_stringify n) (word_stringify w)

and bind_stringify = function
    BindThen(bodies, w) ->
    P.sprintf "%s in %s"
      (String.concat " also "
         (List.map bind_body_stringify bodies))
      (word_stringify w)

and fun_stringify = function
    BQFunction(ds, w)
  | Function(ds, w) ->
    P.sprintf "fun %s = %s"
      (String.concat "; " (List.map arg_def_stringify ds))
      (word_stringify w)

and word_stringify w =
  let _w s n = P.sprintf "(%s of %s)" s n in
  match w with
    WLiteral(pv) -> _w (lit_stringify pv.value_content) "literal"
  | WName(n) -> _w (name_stringify n) "name"
  | WBackquote(bq) -> _w (backquote_stringify bq) "bquote"
  | WSequence(seq) -> _w (seq_stringify seq) "seq"
  | WControl(cs) -> _w (cs_stringify cs) "cs"
  | WFunction(f) -> _w (fun_stringify f) "fun"
  | WBind(b) -> _w (bind_stringify b) "bind"
  | WImport(is) -> "some imports"
  | WExport(ns) -> "some exports"

and words_stringify ws =
  String.concat "/" (List.map word_stringify ws)

and list_stringify lss =
  let rec _strf s ls =
    match ls with
      [] -> s
    | w::ws ->
      _strf (String.concat " | " [(word_stringify w); s]) ws
  in _strf "" lss;;
```

B.5 compiler/compile.ml

Code Listing

```
(* compiler/compile.ml -- Author: Zihang Chen (zc2324) *)
open Switches;;
open Cseg;;
open Common;;
open Stdint;;
open Exp;;
```

```
open Tasm_ast;;
open Config;;
open Scoping;;

let global_fn_id = ref "";;
let global_snippets = ref [];;

type ctx_t = {
  sw: switches; mode: int; (* 1 for push; 2 for pat *)
  is_body: bool; is_backquoted: bool;
  scp_stk: scope_t list; ext_scope_meta: external_scope_t;
};;

let __unique64 = Common.counter ();;
let uniq64 x = Printf.sprintf ":%s-%s" !global_fn_id @@ tu64 @@ __unique64 x;;

let aggregate = List.fold_left (|~~|) cnil;;

let negate_label l = l ^ "!";;

let lmap = List.map;;
let flmap s f = List.map f s;;

let atom_dict = Hashtbl.create 512;;
let atom_repr_tick = Common.counter ();;
Hashtbl.replace atom_dict "false" Uint64.zero;;
Hashtbl.replace atom_dict "true" @@ atom_repr_tick ();;

let mod_uid, _ = name_repr_tick_gen ();;

let name_uid, _ = name_repr_tick_gen ();;
(* 2**64 names should surely be enough. Or I'll say it's more than enough. And
   may cause problem in bytecode generation.
   The reasons that I'm reluctant to change it to int or uint16 are that
   (1) I'm so lazy;
   (2) There isn't int or uint16 in TAsm, there is only uint64, int64 and big_int.
 *)

let ext_scope_tick = Common.counter ();;
ignore (ext_scope_tick ());; (* Module id: 0 for self, 1 for main,
                                2 .. for ext modules. *)

let fun_tick = Common.counter ();;

let (--) x y = Printf.sprintf "%s-%s" x y;;
let (^-) x y = Printf.sprintf "%d-%s" x y;;

let exp_scope:(string, name_t) Hashtbl.t = Hashtbl.create 512;;
let export ctx ns =
  let _export_one n =
    Hashtbl.replace exp_scope n.Ast.name_repr @@ lookup_name ctx.scp_stk n;
    line (EXPORT(ArgLit(VUFixedInt(lookup_name ctx.scp_stk n))))
  in aggregate (List.map _export_one ns);
```

```
60
61  type callback_arg_t = Words of Ast.word list
62                      | Word of Ast.word;;
63  type inst_ctx_t = {pre: (callback_arg_t -> asm);
64                     post: (callback_arg_t -> asm)};;
65  let inst_nil_ctx = {pre = (fun _ -> cnil);
66                      post = (fun _ -> cnil)};;
67
68  let find_closure ctx tree =
69    (* This function is trivial. It literally traverse the given subtree of the AST,
70       mimics all the scoping behavior, and tries to figure out what's in the local
71       scope and what's not.
72       Yes, a crippled version of g_word. *)
73    (* Only find_closure when you finish generating all the code. Because you have
74       to know all the names bound in current scope to know what to capture.
75       OR MAYBE NOT. *)
76    let outer_scope = ctx.scp_stk
77
78    in let table = Hashtbl.create 512
79    in let add_name local_scps pn =
80      let ns = match pn with
81          Ast.NRegular(ns) -> ns
82        | Ast.NTailCall(ns) -> ns
83      in if List.length ns = 1
84      then try if (Uint64.compare
85                     (* Cope with is_DEBUG. *)
86                  (lookup_name local_scps (List.hd ns)) Uint64.zero) <> 0
87      (* Search in local scopes to see if the names exists, e.g.
88         fun A,
89           fun B,
90             bind C D
91             then (C)
92         if we are finding closure about the outmost function,
93         A is local, B is local, C is local, D is nonlocal,
94         or A, B, C appear bound, D appears free.
95
96         See also the comment at Ast.WSequence. *)
97          then (* Do nothing. *) ()
98          else (* Outside of current scope, add to the closure set. *)
99            Hashtbl.replace table
100             ((lookup_name outer_scope (List.hd ns)), Uint64.zero) 1
101       with Exc.NameNotFoundError(_) ->
102         Hashtbl.replace table
103           ((lookup_name outer_scope (List.hd ns)), Uint64.zero) 1
104
105     else (* See num.ml:218 for why we don't need to put ext names in closure.
106             Hashtbl.replace table
107             (lookup_ext_name ctx.ext_scope_meta ns) 1 *)
108       ()
109
110   in let rec __find_in locals is_body =
111        function
112         Ast.WName(pn) -> add_name locals pn
```

```
113         | Ast.WLiteral(pv) ->
114           (match pv.Ast.value_content with
115             Ast.VList(wl) -> List.iter (__find_in locals false) wl
116           | Ast.VTuple(wl) -> List.iter (__find_in locals false) wl
117           | _ -> ()) (* No name references in other kind of literals. *)
118         | Ast.WBackquote(bq) ->
119           (match bq with
120              Ast.BQName(n) -> add_name locals n
121            | Ast.BQSeq(seq) -> __find_in locals false (Ast.WSequence(seq))
122            | Ast.BQValue(v) -> __find_in locals false (Ast.WLiteral(v))
123            | Ast.BQBackquote(b) -> __find_in locals false (Ast.WBackquote(b)))
124         | Ast.WSequence(seq) ->
125           (match seq with
126            (* What if you do
127               fun A,
128                 (B
129                  fun C,
130                    bind B D
131                    also E F
132                    then (C B)
133                  B
134                  E)
135             B should still be in the closure set. So should E. *)
136             Ast.Sequence(ws) ->
137             let new_scope = locals
138             in List.iter (__find_in new_scope false) ws
139            | Ast.SharedSequence(ws) ->
140             (* If it's a shared sequence, do the same as a opt'ed sequence. *)
141             List.iter (__find_in locals false) ws)
142         | Ast.WControl(ctrl) ->
143           (match ctrl with
144              Ast.CtrlSeqIfForm(ifs) ->
145              (match ifs with
146                 Ast.IfGEZ(ib) | Ast.IfGZ(ib) | Ast.IfLEZ(ib)
147               | Ast.IfLZ(ib) | Ast.IfEmpty(ib) | Ast.IfNonEmpty(ib)
148               | Ast.IfEZ(ib) | Ast.IfNEZ(ib) | Ast.IfT(ib) | Ast.IfF(ib) ->
149                 (match ib with
150                    Ast.IfBody(ib1, ib2) ->
151                    __find_in locals true ib1;
152                    __find_in locals true ib2)))
153         | Ast.WFunction(f) ->
154           (match f with
155              Ast.Function(args, body)
156            | Ast.BQFunction(args, body) ->
157              let new_locals = push_scope locals
158              in List.iter (function
159                    Ast.ArgDef(pn) ->
160                    push_name new_locals pn Uint64.one) args;
161              __find_in new_locals true body)
162         | Ast.WBind(Ast.BindThen(bs, bt)) ->
163           let new_locals = push_scope locals
164           in List.iter (function
165                 Ast.BindBody(pn, b) -> push_name new_locals pn Uint64.one;
```

```
                __find_in new_locals false b) bs;
          __find_in new_locals true bt
      | Ast.WImport(_) -> ()
      | Ast.WExport(_) -> ()
  in __find_in [] ctx.is_body tree;
  table;;

let rec g_lit ctx inst_ctx lit =
  match lit.Ast.value_content with
    Ast.VAtom(atom) ->
    let repr =
      (try Hashtbl.find atom_dict atom.Ast.atom_name
       with Not_found ->
         let r = atom_repr_tick ()
         in Hashtbl.add atom_dict atom.Ast.atom_name r; r)
    in (inst_ctx.pre (Word(Ast.WLiteral(lit))))
       |~~| (line (PUSH_LIT(ArgLit(VAtom(repr)))))
       |~~| (inst_ctx.post (Word(Ast.WLiteral(lit))))
  | Ast.VFixedInt(i) ->
    (inst_ctx.pre (Word(Ast.WLiteral(lit))))
    |~~| (line (PUSH_LIT(ArgLit(VFixedInt(i)))))
    |~~| (inst_ctx.post (Word(Ast.WLiteral(lit))))
  | Ast.VUFixedInt(u) ->
    (inst_ctx.pre (Word(Ast.WLiteral(lit))))
    |~~| (line (PUSH_LIT(ArgLit(VUFixedInt(u)))))
    |~~| (inst_ctx.post (Word(Ast.WLiteral(lit))))
  | Ast.VInt(i) ->
    (inst_ctx.pre (Word(Ast.WLiteral(lit))))
    |~~| (line (PUSH_LIT(ArgLit(VInt(i)))))
    |~~| (inst_ctx.post (Word(Ast.WLiteral(lit))))
  | Ast.VFloat(f) ->
    (inst_ctx.pre (Word(Ast.WLiteral(lit))))
    |~~| (line (PUSH_LIT(ArgLit(VFloat(f)))))
    |~~| (inst_ctx.post (Word(Ast.WLiteral(lit))))
  | Ast.VString(s) ->
    (inst_ctx.pre (Word(Ast.WLiteral(lit))))
    |~~| (line (PUSH_LIT(ArgLit(VString(s)))))
    |~~| (inst_ctx.post (Word(Ast.WLiteral(lit))))
  | Ast.VList(wl) ->
    (inst_ctx.pre (Words(wl)))
    |~~| (line (PUSH_LNIL))
    |~~| inst_ctx.post (Words(wl))
    |~~| (let r = flmap wl (g_word ctx inst_nil_ctx)
          in aggregate r)
    |~~| (line END_LIST)
  | Ast.VTuple(wl) ->
    (inst_ctx.pre (Words(wl)))
    |~~| (line (PUSH_TNIL))
    |~~| inst_ctx.post (Words(wl))
    |~~| (let r = flmap wl (g_word ctx inst_nil_ctx)
          in aggregate r)
    |~~| (line END_TUPLE)

```

```
219 and g_import ctx imp =
220   let is_explicit, ss = match imp with Ast.ExplicitImport(ss) -> true, ss
221                                      | Ast.ImplicitImport(ss) -> false, ss
222
223   in let bindings = ref []
224
225   in let push_into_scope ext_scope =
226        let sorted_ext_scope =
227          List.sort (fun x y -> Pervasives.compare (snd x) (snd y))
228          @@ Hashtbl.fold (fun k v acc -> (k, v)::acc) ext_scope []
229        in List.iter (fun x ->
230            let k, v = x
231            in push_name ctx.scp_stk {Ast.name_repr = k}
232              (name_uid k)) sorted_ext_scope
233
234   in let rec find_module mod_str = function
235         path::rest ->
236         let possible_mod_path = Filename.concat path (mod_str ^ ".e")
237         in if BatSys.file_exists possible_mod_path
238         then let ext_scope = open_export possible_mod_path
239           in begin
240             push_ext_scope ctx.ext_scope_meta ext_scope
241               (mod_uid mod_str) mod_str;
242             if not is_explicit
243             then begin
244               bindings := Hashtbl.fold (fun name uid acc ->
245                   ((line (PUSH_NAME(ArgLit(VUFixedInt(uid)),
246                                     ArgLit(VUFixedInt(mod_uid mod_str)))))
247                    |~~| (line (BIND(ArgLit(VUFixedInt(name_uid name)))))::acc)
248                   ext_scope [];
249               push_into_scope ext_scope
250             end else ()
251           end
252
253         else find_module mod_str rest
254       | [] -> failwith (Printf.sprintf "Requested module `%s' not found." mod_str)
255
256   in let () = List.iter (fun x -> find_module x Config.libpaths) ss
257   in List.fold_left (|~~|) cnil
258   @@ lmap (fun x ->
259       (line (PUSH_LIT(ArgLit(VString(x)))))
260       |~~| (line (IMPORT(ArgLit(VUFixedInt(mod_uid x)))))
261       |~~| (if is_explicit
262             then cnil
263             else aggregate !bindings)) ss
264
265 and g_bind ctx =
266   let new_scps = push_scope ctx.scp_stk
267   in let _g_bind_body =
268     function
269       Ast.BindBody(pn, b) ->
270       (push_name new_scps pn @@ name_uid pn.Ast.name_repr);
271       (cnil
```

```
272         |~~| (g_word {ctx with is_body = false;
273                                is_backquoted = false;
274                                scp_stk = new_scps}
275                 {inst_nil_ctx
276                  with post =
277                         fun _ -> line (BIND(ArgLit(VUFixedInt(
278                             lookup_name new_scps pn))))}
279                 b))
280       (* Cannot do compile-time stack indexing, i.e. giving names the stack index,
281          because you don't know what will come out of the calculation in `b'. *)
282
283   in function
284       Ast.BindThen(bs, b) ->
285       let bind_inst = aggregate (lmap _g_bind_body bs)
286       in let then_inst = g_word {ctx with is_body = true; is_backquoted = false;
287                                           scp_stk = new_scps}
288              inst_nil_ctx b
289       in (line PUSH_SCOPE) |~~| bind_inst |~~| then_inst |~~| (line POP_SCOPE)
290
291 and g_name ctx inst_ctx n =
292   let get_args ns = if List.length ns = 1
293     then ArgLit(VUFixedInt(lookup_name ctx.scp_stk (List.hd ns))),
294          ArgLit(VUFixedInt(Uint64.zero))
295     else let n_uid, es_uid = lookup_ext_name ctx.ext_scope_meta ns
296       in ArgLit(VUFixedInt(n_uid)), ArgLit(VUFixedInt(es_uid))
297   in let inst = match n with
298         Ast.NRegular(ns) ->
299         let arg1, arg2 = get_args ns
300         in if ctx.is_backquoted
301         then line (PUSH_NAME(arg1, arg2))
302         else line (EVAL_AND_PUSH(arg1, arg2))
303
304       | Ast.NTailCall(ns) ->
305         let arg1, arg2 = get_args ns
306         in if ctx.is_backquoted
307         then line (PUSH_NAME(arg1, arg2)) (* This is a weird case. *)
308         else line (EVAL_TAIL(arg1, arg2))
309   in cnil
310      |~~| inst_ctx.pre (Word(Ast.WName(n)))
311      |~~| inst
312      |~~| inst_ctx.post (Word(Ast.WName(n)))
313
314 and g_ctrl ctx =
315   function
316     Ast.CtrlSeqIfForm(i) -> g_if ctx i
317
318 and g_if ctx inst =
319   let _UID = uniq64 ()
320   in let brancht_id = _UID
321   in let branchf_id = negate_label _UID
322   in let is_hungry = sw_hungry_if ctx.sw
323
324   in let if_body_ = function
```

```
325           Ast.IfBody(bt, bf) ->
326           let brancht = g_word {ctx with is_body = true;
327                                          is_backquoted = false} inst_nil_ctx bt
328           in let branchf = g_word {ctx with is_body = true;
329                                            is_backquoted = false} inst_nil_ctx bf
330           in let if_end_id = Printf.sprintf "%s-end" _UID
331
332           in (put_label [brancht_id]) |~~| brancht
333              |~~| (line (JUMP(ArgLabel(Label(if_end_id)))))
334              |~~| (put_label [branchf_id]) |~~| branchf
335              |~~| (put_label [if_end_id])
336
337     in let _g_body ib i =
338          cnil |~~| (line i) |~~| (if_body_ ib)
339
340     in let parse = let lbl = ArgLabel(Label(branchf_id))
341          in function
342              Ast.IfGEZ(ib) -> ib, (if is_hungry then HJLZ(lbl) else JLZ(lbl))
343            | Ast.IfGZ(ib) -> ib, if is_hungry then HJLEZ(lbl) else JLEZ(lbl)
344            | Ast.IfLEZ(ib) -> ib, if is_hungry then HJGZ(lbl) else JGZ(lbl)
345            | Ast.IfLZ(ib) -> ib, if is_hungry then HJGEZ(lbl) else JGEZ(lbl)
346            | Ast.IfEZ(ib) -> ib, if is_hungry then HJNEZ(lbl) else JNEZ(lbl)
347            | Ast.IfNEZ(ib) -> ib, if is_hungry then HJEZ(lbl) else JEZ(lbl)
348            | Ast.IfT(ib) -> ib, if is_hungry then HJF(lbl) else JF(lbl)
349            | Ast.IfF(ib) -> ib, if is_hungry then HJT(lbl) else JT(lbl)
350            | Ast.IfEmpty(ib) -> ib, JNE(lbl)
351            | Ast.IfNonEmpty(ib) -> ib, if is_hungry then HJE(lbl) else JE(lbl)
352
353     in let body, ins = parse inst
354     in _g_body body ins
355
356   and g_backquote ctx inst_ctx =
357     let new_ctx = {ctx with is_backquoted = true}
358     in function
359       Ast.BQValue(pv) -> g_lit new_ctx inst_ctx pv
360     | Ast.BQName(n) -> g_name new_ctx inst_ctx n
361     | Ast.BQSeq(seq) -> g_seq new_ctx inst_ctx seq
362     | Ast.BQBackquote(bq) -> g_backquote ctx inst_ctx bq
363
364   and g_seq ctx inst_ctx seq =
365     let _UID = uniq64 ()
366
367     in let seq_st_id = Printf.sprintf "%s-st" _UID
368     in let seq_end_id = Printf.sprintf "%s-end" _UID
369     in let seq_real_end_id = if _UID = "na" then ""
370          else Printf.sprintf "%s-real-end" _UID
371
372     in let is_shared = match seq with
373           Ast.Sequence(_) -> false
374         | Ast.SharedSequence(_) -> true
375
376     in let opt_cs what =
377          if sw_opt_seq ctx.sw
```

```
        then if ctx.is_body
        (* Optimize out some of the boilerplate code based on the current
           switches settings. *)
          then cnil
          else what
        else what

    in let seq_preamble =
        let _x = if not is_shared
          then (put_label [seq_st_id])
               |~~| (line PUSH_STACK)
               |~~| (line PUSH_SCOPE)
          else put_label [seq_st_id]
        in opt_cs _x

    in let seq_postamble =
        let _x = if not is_shared
          then (put_label [seq_end_id])
               |~~| (line POP_SCOPE)
               |~~| (line RET)
          else (put_label [seq_end_id])
               |~~| (line SHARED_RET)
        in (opt_cs (_x |~~| (put_label [seq_real_end_id])))

    in let body, scp_stk = match seq with
         Ast.Sequence(s) -> s, ctx.scp_stk
       | Ast.SharedSequence(s) -> s, ctx.scp_stk

    in let lead_inst = if ctx.is_backquoted
        then (line (PUSH_FUN(ArgLabel(Label(seq_st_id)))))
        else (line (CALL(ArgLabel(Label(seq_st_id)))))
    in let body_insts = aggregate (lmap (g_word {ctx with is_body = false;
                                                          is_backquoted = false;
                                                          scp_stk = scp_stk}
                                            inst_nil_ctx) body)
    in let seq_insts =
        cnil
        |~~| seq_preamble
        |~~| (if ctx.is_backquoted
              then (line INSTALL)
              else cnil)
        |~~| body_insts
        |~~| seq_postamble

    in let closure = if ctx.is_backquoted
        then find_closure ctx (Ast.WSequence(seq))
        (* As it turns out, ctx.scp_stk can totally represent the state of
           the scope stack before we enter this sequence context. *)
        else Hashtbl.create 1
    in let closure_insts =
        (Hashtbl.fold (fun k v acc ->
            let nid, esid = k
            in acc |~~| line (CLOSURE(ArgLit(VUFixedInt(nid)))))
```

```
431             closure cnil)
432
433   in let () = if ctx.is_body
434        then ()
435        else global_snippets := seq_insts::(!global_snippets)
436
437   in let body_main = if ctx.is_body
438        then seq_insts
439        else cnil
440        (* We cannot put shared seq_insts in global_snippets like functions,
441           because shared sequences don't have return instruction to go back
442           to where we began. *)
443
444   in inst_ctx.pre (Word(Ast.WSequence(seq)))
445      |~~| (opt_cs lead_inst)
446      |~~| inst_ctx.post (Word(Ast.WSequence(seq)))
447      |~~| closure_insts
448      |~~| body_main
449
450 and g_fun ctx inst_ctx fun_ =
451   let _UID = uniq64 ()
452   in let st_label = _UID -- "st"
453   in let end_label = _UID -- "end"
454   in let real_end_label = _UID -- "real-end"
455
456   in let preamble = (put_label [st_label])
457                     |~~| (line (if sw_share_stack ctx.sw
458                                 then SHARE_STACK
459                                 else PUSH_STACK))
460                     |~~| (line PUSH_SCOPE)
461   in let scp_stk = push_scope ctx.scp_stk
462          (* A function invocation automatically pushes a scope (for its arguments),
463             so no need to explicitly write a push-scope. *)
464   in let _g_arg_def = function
465         Ast.ArgDef(pn) ->
466         push_name scp_stk pn @@ name_uid pn.Ast.name_repr;
467         line (FUN_ARG(ArgLit(VUFixedInt(lookup_name scp_stk pn))))
468
469   in let _g_fun arg_defs body is_backquoted =
470        let closure = if is_backquoted
471          then find_closure ctx (Ast.WFunction(fun_))
472          else Hashtbl.create 1
473        in let closure_insts =
474             Hashtbl.fold (fun k v acc ->
475                 let nid, esid = k
476                 in acc |~~| line (CLOSURE(ArgLit(VUFixedInt(nid)))))
477               closure cnil
478
479        in let fun_inst = if is_backquoted
480             then PUSH_FUN(ArgLabel(Label(st_label)))
481             else CALL(ArgLabel(Label(st_label)))
482
483        in let main = cnil
```

```
                    |~~| inst_ctx.pre (Word(Ast.WFunction(fun_)))
                    |~~| (line fun_inst)
                    |~~| inst_ctx.post (Word(Ast.WFunction(fun_)))
                    |~~| closure_insts

        in let fun_args = aggregate @@ List.rev @@ lmap _g_arg_def arg_defs
        in let body_inst = g_word {ctx with is_body = true;
                                            scp_stk = scp_stk;
                                            is_backquoted = false}
               inst_nil_ctx body
        in let snippet = cnil
                         |~~| preamble
                         |~~| (line INSTALL)
                         |~~| fun_args
                         |~~| (line SWEEP)
                         |~~| body_inst
                         |~~| (cline [end_label] (Some(POP_SCOPE)))
                         |~~| (line RET)
                         |~~| (put_label [real_end_label])
        in global_snippets := snippet::(!global_snippets);
        main
   in match fun_ with
       Ast.BQFunction(arg_defs, body) -> _g_fun arg_defs body true
     | Ast.Function(arg_defs, body) -> _g_fun arg_defs body false

and g_word ctx inst_ctx = function
    Ast.WLiteral(pv) -> g_lit ctx inst_ctx pv
  | Ast.WName(n) -> g_name ctx inst_ctx n
  | Ast.WBackquote(bq) -> g_backquote ctx inst_ctx bq
  | Ast.WSequence(seq) -> g_seq ctx inst_ctx seq
  | Ast.WControl(ctrl) -> g_ctrl ctx ctrl
  | Ast.WFunction(f) -> g_fun ctx inst_ctx f
  | Ast.WBind(b) -> g_bind ctx b
  | Ast.WImport(is) -> g_import ctx is
  | Ast.WExport(ns) -> export ctx ns

let compile cst fn sw =
  let scope_stack_init = push_scope []

  in global_fn_id := fn;
  let aLL_THE_REST_OF_THAT =
       lmap (g_word {sw = sw; mode = 1; scp_stk = scope_stack_init;
                     is_body = true; is_backquoted = false;
                     ext_scope_meta = ExtScope(Hashtbl.create 512,
                                               Hashtbl.create 512,
                                               Uint64.zero)}
              inst_nil_ctx)
         (match cst with
            Ast.Sentence(ws) -> ws)
  in (cnil
      |~~| (line PUSH_STACK)
      |~~| (line PUSH_SCOPE)
      |~~| (aggregate aLL_THE_REST_OF_THAT)
```

```
537         |~~| (line TERMINATE)
538         |~~| (aggregate !global_snippets)),
539         exp_scope
```

B.6 compiler/config.ml

Code Listing

```
1  (* compiler/config.ml -- Author: Zihang Chen (zc2324) *)
2
3  let libpaths = [Sys.getcwd (); "/Users/wo/ocaml-wksp/Towel/src/towelibs"];;
```

B.7 compiler/cseg.ml

Code Listing

```
1  (* compiler/cseg.ml -- Author: Zihang Chen (zc2324) *)
2  open Tasm_ast;;
3
4  let make_labels =
5    let labelize x = Label(x)
6    in List.map labelize;;
7
8  let cnil = Asm([CLine([], None)]);;
9  let line x = Asm([Line([], x)]);;
10 let lline lbs x = Asm([Line(make_labels lbs, x)]);;
11 let cline lbs inst = Asm([CLine(make_labels lbs, inst)]);;
12 let put_label lbs = cline lbs None;;
13
14 let (|~~|) a1 a2 =
15   let lr = List.rev
16   in let lh = List.hd
17   in let lt = List.tl
18   in let lfirstn xs = xs |> lr |> lt |> lr
19   in match a1, a2 with
20     Asm([]), Asm(_) ->
21     a2 (* If a1 is empty, who cares about a1? *)
22   | Asm(ls1), Asm([]) ->
23     a1
24   | Asm(ls1), Asm(ls2) ->
25     let last1 = lh (lr ls1)
26     in let first2 = lh ls2
27     in let reject_empty = List.filter
28            (fun x -> (Pervasives.compare x (CLine([], None))) <> 0)
29     in let content = match last1, first2 with
30           CLine([], None), CLine([], None) ->
31           (* 0 + 0 = 0 *)
32           (lfirstn ls1) @ (lt ls2)
33         | Line(_, _), Line(_, _) ->
34           (* non-0 + non-0 = non-0 *)
35           ls1 @ ls2
36         | _, CLine([], None) ->
```

```
            (* non-0 + 0 = non-0 *)
            ls1 @ (lt ls2)
          | CLine([], None), _ ->
            (* 0 + non-0 = non-0 *)
            (lfirstn ls1) @ ls2
          | Line([], inst1), CLine(lbs2, Some(inst2)) ->
            (* Transform all the CLine into Line. *)
            ls1 @ ((Line(lbs2, inst2))::(lt ls2))
          | Line(_, _), CLine([], Some(inst2)) ->
            (* Can't merge. *)
            ls1 @ ((Line([], inst2))::(lt ls2))
          | Line(_, _), CLine(lbs2, Some(inst2)) ->
            (* Can't merge. *)
            ls1 @ ((Line(lbs2, inst2))::(lt ls2))
          | Line(lbs1, inst1), CLine(lbs2, None) ->
            (* Can't merge. Can't transform. *)
            ls1 @ ls2
          | CLine([], Some(inst1)), Line(_, _) ->
            (lfirstn ls1) @ [Line([], inst1)] @ ls2
          | CLine(lbs1, None), Line([], inst2) ->
            (lfirstn ls1) @ [Line(lbs1, inst2)] @ (lt ls2)
          | CLine(lbs1, None), Line(lbs2, inst2) ->
            (lfirstn ls1) @ [Line(lbs1 @ lbs2, inst2)] @ (lt ls2)
          | CLine(lbs1, None), CLine(lbs2, None) ->
            (lfirstn ls1) @ [CLine(lbs1 @ lbs2, None)] @ (lt ls2)
          | CLine(lbs1, None), CLine(lbs2, Some(inst2)) ->
            (lfirstn ls1) @ [Line(lbs1 @ lbs2, inst2)] @ (lt ls2)
          | CLine(lbs1, Some(inst1)), Line(_, _) ->
            (lfirstn ls1) @ [Line(lbs1, inst1)] @ ls2
          | CLine(lbs1, Some(inst1)), CLine(lbs2, Some(inst2)) ->
            (lfirstn ls1) @ [Line(lbs1, inst1); Line(lbs2, inst2)] @ (lt ls2)
          | CLine(lbs1, Some(inst1)), CLine(lbs2, None) ->
            (lfirstn ls1) @ [Line(lbs1 @ lbs2, inst1)] @ (lt ls2)
      in Asm(reject_empty content)
```

B.8 compiler/exc.ml

Code Listing

```
(* compiler/exc.ml -- Author: Zihang Chen (zc2324) *)

exception LexicalError of (string * int * int);;
exception SyntacticError of (string * int * int * int);;
exception NameNotFoundError of string;;
exception CorruptedScope of string;;

exception TypeError;;

let err s loc stofs eofs = raise (SyntacticError(s,
                                                 loc.Lexing.pos_lnum,
                                                 stofs,
                                                 eofs));;
```

B.9 compiler/exp.ml

Code Listing

```
(* compiler/exp.ml -- Author: Zihang Chen (zc2324) *)
open Stdint;;
open Ast;;
open Scoping;;

(* ========================================
     Export file parser
   ======================================== *)

let exp_filename fn =
  String.concat "." [fn; "e"];;

let parse cst =
  let parse_one acc =
    function
      WSequence(Sequence(ws)) ->
      let name =
        match List.hd ws with
          WName(NRegular(ns)) -> List.hd ns
        | _ -> failwith "WTF a non name word?"

      in let idx =
           match List.hd (List.tl ws) with
             WLiteral(x) ->
             (match x.value_content with
                VUFixedInt(u) -> u
              | _ -> failwith "WTF a non ufixedint as index?")
           | _ -> failwith "WTF a non literal as index?"

      in Hashtbl.replace acc name.name_repr idx; acc
    (* I apologize for so much nested match expressions. :( *)
    | _ -> failwith "Unrecognized structure in .E file."

  in match cst with
    Sentence(ws) -> List.fold_left parse_one
                      (Hashtbl.create 512)
                      ws

let open_export impn =
  let lexbuf = Lexing.from_channel @@ Pervasives.open_in impn
  in let cst = Parser.sentence Scanner.token lexbuf
  in parse cst;;
```

B.10 compiler/parser.mly

Code Listing

```
(* compiler/parser.mly -- Author: Zihang Chen (zc2324) *)
```

```
%{
open Ast
open Common
open Exc

%}

%token IFGEZ IFGZ IFLEZ IFLZ IFE IFNE IFEZ IFNEZ IFT IFF
%token FUNCTION BIND ALSO THEN AT IMPORT EXPORT
%token SLASH BQUOTE COMMA LAMBDA
%token LBRACKET RBRACKET LPAREN RPAREN LBRACE RBRACE

%token <Ast.atom> ATOM
%token <string> STRING
%token <Ast.pvalue> LITERAL
%token <Ast.pname> NAME
%token <Ast.terminator> TERMINATOR

%start sentence
%type <Ast.sentence> sentence
%%

lit_list:
  LBRACKET list(word) RBRACKET { VList($2) }
| LBRACKET list(word) error {
    err "expected a right bracket for list literal"
      $startpos($3) $startofs($1) $endofs($3)
  }

lit_tuple:
  LBRACKET SLASH list(word) RBRACKET { VTuple($3) }

literal:
  LITERAL { $1 }
| STRING { {value_content = VString($1)} }
| ATOM { {value_content = VAtom($1) } }
| lit_list { {value_content = $1} }
| lit_tuple { {value_content = $1} }

backquote:
  literal BQUOTE { BQValue($1) }
| name BQUOTE { BQName($1) }
| sequence BQUOTE { BQSeq($1) }
| backquote BQUOTE { BQBackquote($1) }
| LBRACE list(word) RBRACE { BQSeq(SharedSequence($2)) }

arg_def:
  NAME { ArgDef($1) }

if_sform:
  IFGEZ word COMMA word { IfGEZ(IfBody($2, $4)) }
| IFGZ word COMMA word { IfGZ(IfBody($2, $4)) }
| IFLEZ word COMMA word { IfLEZ(IfBody($2, $4)) }
```

```
  | IFLZ word COMMA word { IfLZ(IfBody($2, $4)) }
  | IFE word COMMA word { IfEmpty(IfBody($2, $4)) }
  | IFNE word COMMA word { IfNonEmpty(IfBody($2, $4)) }
  | IFEZ word COMMA word { IfEZ(IfBody($2, $4)) }
  | IFNEZ word COMMA word { IfNEZ(IfBody($2, $4)) }
  | IFT word COMMA word { IfT(IfBody($2, $4)) }
  | IFF word COMMA word { IfF(IfBody($2, $4)) }
  | IFGEZ word error {
      err "expected a comma for else branch"
      $startpos($3) $startofs($1) $endofs($3)
    }
  | IFGZ word error {
      err "expected a comma for else branch"
      $startpos($3) $startofs($1) $endofs($3)
    }
  | IFLEZ word error {
      err "expected a comma for else branch"
      $startpos($3) $startofs($1) $endofs($3)
    }
  | IFLZ word error {
      err "expected a comma for else branch"
      $startpos($3) $startofs($1) $endofs($3)
    }
  | IFE word error {
      err "expected a comma for else branch"
      $startpos($3) $startofs($1) $endofs($3)
    }
  | IFNE word error {
      err "expected a comma for else branch"
      $startpos($3) $startofs($1) $endofs($3)
    }
  | IFEZ word error {
      err "expected a comma for else branch"
      $startpos($3) $startofs($1) $endofs($3)
    }
  | IFNEZ word error {
      err "expected a comma for else branch"
      $startpos($3) $startofs($1) $endofs($3)
    }
  | IFT word error {
      err "expected a comma for else branch"
      $startpos($3) $startofs($1) $endofs($3)
    }
  | IFF word error {
      err "expected a comma for else branch"
      $startpos($3) $startofs($1) $endofs($3)
    }
  | IFGEZ word COMMA error {
      err "unexpected form for else branch"
      $startpos($3) $startofs($1) $endofs($3)
    }
  | IFGZ word COMMA error {
      err "unexpected form for else branch"
```

```
    $startpos($3) $startofs($1) $endofs($3)
  }
| IFLEZ word COMMA error {
    err "unexpected form for else branch"
    $startpos($3) $startofs($1) $endofs($3)
  }
| IFLZ word COMMA error {
    err "unexpected form for else branch"
    $startpos($3) $startofs($1) $endofs($3)
  }
| IFEZ word COMMA error {
    err "unexpected form for else branch"
    $startpos($3) $startofs($1) $endofs($3)
  }
| IFNEZ word COMMA error {
    err "unexpected form for else branch"
    $startpos($3) $startofs($1) $endofs($3)
  }
| IFE word COMMA error {
    err "unexpected form for else branch"
    $startpos($3) $startofs($1) $endofs($3)
  }
| IFNE word COMMA error {
    err "unexpected form for else branch"
    $startpos($3) $startofs($1) $endofs($3)
  }
| IFT word COMMA error {
    err "unexpected form for else branch"
    $startpos($3) $startofs($1) $endofs($3)
  }
| IFF word COMMA error {
    err "unexpected form for else branch"
    $startpos($3) $startofs($1) $endofs($3)
  }
| IFGEZ error {
    err "unexpected form for if branch"
    $startpos($2) $startofs($1) $endofs($2)
  }
| IFGZ error {
    err "unexpected form for if branch"
    $startpos($2) $startofs($1) $endofs($2)
  }
| IFLEZ error {
    err "unexpected form for if branch"
    $startpos($2) $startofs($1) $endofs($2)
  }
| IFLZ error {
    err "unexpected form for if branch"
    $startpos($2) $startofs($1) $endofs($2)
  }
| IFE error {
    err "unexpected form for if branch"
    $startpos($2) $startofs($1) $endofs($2)
```

```
161     }
162   | IFNE error {
163       err "unexpected form for if branch"
164       $startpos($2) $startofs($1) $endofs($2)
165     }
166   | IFT error {
167       err "unexpected form for if branch"
168       $startpos($2) $startofs($1) $endofs($2)
169     }
170   | IFF error {
171       err "unexpected form for if branch"
172       $startpos($2) $startofs($1) $endofs($2)
173     }
174   | IFEZ error {
175       err "unexpected form for if branch"
176       $startpos($2) $startofs($1) $endofs($2)
177     }
178   | IFNEZ error {
179       err "unexpected form for if branch"
180       $startpos($2) $startofs($1) $endofs($2)
181     }
182
183   control_sequence:
184     if_sform { CtrlSeqIfForm($1) }
185
186   name:
187     separated_nonempty_list(SLASH, NAME) { NRegular($1) }
188   | name AT { match $1 with
189         NRegular(x) -> NTailCall(x)
190       | NTailCall(_) as y -> y
191     }
192   | NAME SLASH error {
193       err "expected a name (possibly with a namespace reference)"
194       $startpos($3) $startofs($1) $endofs($3)
195     }
196
197   function_:
198     FUNCTION list(arg_def) COMMA word { Function($2, $4) }
199   | FUNCTION BQUOTE list(arg_def) COMMA word { BQFunction($3, $5) }
200   | LAMBDA list(arg_def) COMMA word { BQFunction($2, $4) }
201   | FUNCTION list(arg_def) error {
202       err "expected a comma for function special form"
203       $startpos($3) $startofs($1) $endofs($3)
204     }
205
206   bind_body:
207     NAME word { BindBody($1, $2) }
208
209   bind_sform:
210     BIND separated_nonempty_list(ALSO, bind_body) THEN word {
211       BindThen($2, $4)
212     }
213   | BIND separated_nonempty_list(ALSO, bind_body) error {
```

```
    err "expected \"then\" for bind-then special form"
    $startpos($3) $startofs($1) $endofs($3)
  }
| BIND separated_nonempty_list(ALSO, bind_body) THEN error {
    err "expected a form for bind-then special form"
    $startpos($4) $startofs($1) $endofs($4)
  }
| BIND error {
    err "expected a name" $startpos($2) $startofs($1) $endofs($2)
  }

import:
  IMPORT list(STRING) SLASH {
    ExplicitImport($2)
  }
| IMPORT list(STRING) AT {
    ImplicitImport($2)
  }

export:
  EXPORT list(NAME) AT {
    $2
  }

word:
  backquote { WBackquote($1) }
| sequence { WSequence($1) }
| literal { WLiteral($1) }
| control_sequence { WControl($1) }
| function_ { WFunction($1) }
| bind_sform { WBind($1) }
| import { WImport($1) }
| export { WExport($1) }
| name { WName($1) }

sequence:
  LPAREN list(word) RPAREN { Sequence($2) }
| LPAREN AT list(word) RPAREN { SharedSequence($3) }
| LPAREN AT error {
    err "expected a list of words for shared sequence"
    $startpos($3) $startofs($1) $endofs($3)
  }
| LPAREN list(word) error {
    err "expected right parenthesis for sequence"
    $startpos($3) $startofs($1) $endofs($3)
  }

sentence:
  list(word) TERMINATOR { Sentence($1) }
| list(word) error {
    err "expected a terminator"
    $startpos($2) $startofs($2) $endofs($2)
  }
```

B.11 compiler/scanner.mll

Code Listing

```
(* compiler/scanner.mll -- Author: Zihang Chen (zc2324) *)
{
open Ast
open Parser
open Common
open Lexing
open Exc
open Stdint
open Batteries

let strip_mod s =
  let len = String.length s
  in String.sub s 0 (len - 1)

let strip_sign s =
  let len = String.length s
  in let first = String.sub s 0 1
  in if (first = "+") || (first = "-")
  then String.sub s 1 @@ len - 1
  else s

let unquote s =
  let rec _unq output state = function
      ch::rest ->
      if ch <> '\\'
      then if state = 0
        then (BatIO.write output ch;
              _unq output 0 rest)
        else (BatIO.write output
                (match ch with
                   'n' -> '\n'
                 | 't' -> '\t'
                 | 'b' -> '\b'
                 | '\'' -> '\''
                 | '\\' -> '\\'
                 | _ -> failwith "Illegal escape sequence.");
              _unq output 0 rest)
      else _unq output 1 rest
    | [] -> BatIO.close_out output
  in let out = BatIO.output_string ()
  in _unq out 0 (String.to_list s)
}

let _WHITESPACE = [' ' '\t']
let _NEWLINE = '\n' | '\r' | "\r\n"
let _SQUOTE = '\''
let _DQUOTE = '"'
let _BQUOTE = '`'
let _COMMA = ','
```

```
let _LPAREN = '('
let _RPAREN = ')'
let _LBRACKET = '['
let _RBRACKET = ']'
let _SLASH = '\\'
let _AT = '@'
let _LBRACE = '{'
let _RBRACE = '}'

let string_char = [^ '\\' '\'']
let string_esc_charseq = '\\' ['\'' '\\' 'n' 'r' 'b' ' ' 't']
let string_item = string_char | string_esc_charseq
let string_lit = _SQUOTE string_item* _SQUOTE
(* From python lexical analysis
   https://docs.python.org/3/reference/lexical_analysis.html#string-and-bytes-literals*)

let alpha = ['a'-'z']
let reserved_char = [',' '.' '\'' '\\' '`' '@'
                     '(' ')' '[' ']' '{' '}'
                     ' ' '\t' '\n' '\r']
let common_valid_char = ['~' '!' '#' '$' '%' '^' '&' '*' '-' '_' '+' '=' '.'
                         '|' ':' '<' '>' '?' '/' 'a'-'z' 'A'-'Z' '0'-'9' ';']
let common_valid_char_no_digits =
  ['~' '!' '#' '$' '%' '^' '&' '*' '-' '_' '+' '=' '.' ';'
   '|' ':' '<' '>' '?' '/' 'a'-'z' 'A'-'Z']

let valid_upper_char = ['~' '!' '#' '$' '%' '^' '&' '*' '_' '=' '.' ';'
                        '|' ':' '<' '>' '?' '/' 'A'-'Z']

let digit = ['0'-'9']
let hexdigit = ['0'-'9' 'a'-'f' 'A'-'F']
let bindigit = ['0' '1']
let signed = ['+' '-']
let _int_lit = (("0d"? digit+)
              | ("0x" hexdigit+)
              | ("0b" bindigit+))
let fint_lit = signed? _int_lit
let int_lit = signed? digit+ ['L' 'l']
let ufint_lit = '+'? _int_lit ['U' 'u']

let dot = '.'
let int = digit+
let frac = digit+
let exp = 'e' signed? int
let dot_float = ((int dot) | (dot frac) | (int dot frac)) exp?
let exp_float = int (dot frac)? exp
let float_lit = signed? (dot_float | exp_float)

let name = (valid_upper_char common_valid_char*)
         | ('+' common_valid_char_no_digits?)
         | ('+' common_valid_char_no_digits common_valid_char*)
         | ('-' common_valid_char_no_digits?)
         | ('-' common_valid_char_no_digits common_valid_char*)
```

```
let atom_lit = alpha common_valid_char*


rule token = parse
| _WHITESPACE+ { token lexbuf }
| _NEWLINE { Lexing.new_line lexbuf; token lexbuf }
| _BQUOTE { BQUOTE }
| _COMMA { COMMA }
| _LPAREN { LPAREN }
| _RPAREN { RPAREN }
| _LBRACKET { LBRACKET }
| _RBRACKET { RBRACKET }
| _SLASH { SLASH }
| _AT { AT }
| _LBRACE { LBRACE }
| _RBRACE { RBRACE }

| "if>=0" { IFGEZ }
| "if>0" { IFGZ }
| "if<=0" { IFLEZ }
| "if<0" { IFLZ }
| "if=0" { IFEZ }
| "if~0" { IFNEZ }
| "ift" { IFT }
| "iff" { IFF }
| "ife" { IFE }
| "ifne" { IFNE }
| "bind" { BIND }
| "also" { ALSO }
| "then" { THEN }
| "fun" { FUNCTION }
| "import" { IMPORT }
| "export" { EXPORT }
| ",\\" { LAMBDA }

| eof { TERMINATOR(Ast.EOF) }

| _DQUOTE { comment lexbuf } (* comments *)

              (* literals start here *)
| atom_lit as a {
    ATOM({atom_name = a;
          atom_repr = 1});
  }
| string_lit as str {
    STRING(unquote(String.sub str 1 (String.length str - 2)))
  }
| fint_lit as i {
    LITERAL({value_content = VFixedInt(Int64.of_string i)})
  }
| int_lit as i {
    LITERAL({value_content = VInt(Big_int.big_int_of_string @@ strip_mod i)})
  }
```

```
  | ufint_lit as i {
      LITERAL({value_content = VUFixedInt(Uint64.of_string
                                          (i |> strip_sign |> strip_mod))})
    }
  | float_lit as f {
      LITERAL({value_content = VFloat(float_of_string f)})
    }

  | name as n {
      NAME({name_repr = n})
    }

  | _ as s {
      raise (LexicalError
               (Printf.sprintf "unexpected character `%c'" s,
                lexbuf.lex_curr_p.pos_lnum,
                lexbuf.lex_curr_p.pos_bol))
    }

and comment = parse
    _DQUOTE { token lexbuf }
  | _ { comment lexbuf }
```

B.12 compiler/scoping.ml

Code Listing

```
(* compiler/scoping.ml -- Author: Zihang Chen (zc2324) *)
open Ast;;
open Stdint;;

(* ========================================
   Scoping utilities for both Towel compiler

   + A Towel module consists of different scopes, and as functions
     being invoked, new scopes along with the functions are pushed/popped
     onto the scope stack.
   + A scope is randomized a hashtable (because it's mutable in OCaml),
     basic operations consists of adding new names (including overwriting),
     looking up names, and maybe some metaprogramming infrastructures.
   ======================================== *)

let is_DEBUG = ref false;;

type name_t = uint64;;
type scope_t = (string, name_t) Hashtbl.t;;

type external_scope_t = ExtScope of (string, external_scope_t) Hashtbl.t (* path table *)
                                  * (string, name_t) Hashtbl.t       (* scope *)
                                  * uint64                           (* id *)

let print_ht ht = print_string "ht:\n";
```

```
    Hashtbl.iter (fun k v -> Printf.printf "%s -> %s\n" k (Uint64.to_string v)) ht;
  print_string "-----\n";;

let print_es es = print_string "es:\n";
  Hashtbl.iter (fun k v -> Printf.printf "%s -> %s\n" k "es")
    (match es with ExtScope(p, _, _) -> p);
  print_ht (match es with ExtScope(_, p, _) -> p);
  print_string "-----\n";;

let name_repr_tick_gen () =
  let table = Hashtbl.create 128
  in let tick = Common.counter ()
  in (fun name -> try
         Hashtbl.find table name
       with Not_found ->
         let x = tick ()
         in Hashtbl.replace table name x;
         x),
     (fun name -> Hashtbl.find table name);;

let push_ext_scope meta ext_scope uid possible_mod_path =
  let rec _push_e_s cur mod_path =
    try let parent, rest = BatString.split Filename.dir_sep mod_path
      in match cur with
        ExtScope(package_path, _, _) ->
        if Hashtbl.mem package_path parent
        then _push_e_s (Hashtbl.find package_path parent) rest
        else Hashtbl.replace package_path (BatString.capitalize parent)
            (ExtScope(Hashtbl.create 512, Hashtbl.create 512, Uint64.zero));
        _push_e_s (Hashtbl.find package_path parent) rest

    with Not_found ->
    match cur with
    ExtScope(paths, _, _) ->
      Hashtbl.replace paths (BatString.capitalize mod_path)
        (ExtScope(Hashtbl.create 512, ext_scope, uid))
  in _push_e_s meta possible_mod_path;;

let lookup_ext_name meta ns =
  let rec _lookup cur = function
      [] -> failwith "Not found."
    | n::[] -> (match cur with
          ExtScope(_, ext_scope, uid) -> Hashtbl.find ext_scope n.name_repr, uid)
    | name::rest -> (match cur with
          ExtScope(p, _, _) -> _lookup (Hashtbl.find p name.name_repr) rest)

  in _lookup meta (List.rev ns)

let init_scope_stack = [];;

let push_scope ?ctx_name:(c="unnamed ctx") scp_stk =
  (Hashtbl.create ~random:true 512)::scp_stk;;

```

```
let pop_scope scp_stk =
  match scp_stk with
    _::rest -> rest
  | [] -> [];;

let push_name scp_stk name value =
  Hashtbl.replace (List.hd scp_stk) name.name_repr value;;

let print_scp_stk scp_stk = print_string "scope stack:\n";
    List.iter (fun h -> print_ht h) scp_stk;
  print_string "_____\n";;

let pop_name scp_stk name =
  Hashtbl.remove (List.hd scp_stk) name.name_repr;;

let rec lookup_name scp_stk name =
  match scp_stk with
    [] -> if !is_DEBUG (* culture shock!! *)
    then Uint64.zero
    else raise (Exc.NameNotFoundError
                  (Printf.sprintf
                     "requested name `%s' not found" name.name_repr))
  | scp::rest ->
    try
      let vs = Hashtbl.find_all scp name.name_repr in
      match vs with
        [] -> lookup_name rest name
      | v::[] -> v
      | v::vs -> raise (Exc.CorruptedScope
                          (Printf.sprintf "scope %s is corrupted"
                           "some scope"))
    with Not_found -> lookup_name rest name
```

B.13 compiler/switches.ml

Code Listing

```
(* compiler/switches.ml -- Author: Zihang Chen (zc2324) *)
open Batteries;;

type switches =
    CompilerSwitches of bool * (* hungry *)
                        bool * (* share stack *)
                        bool;; (* opt seq *)

let sw_hungry_if = function CompilerSwitches(b, _, _) -> b;;
let sw_share_stack = function CompilerSwitches(_, b, _) -> b;;
let sw_opt_seq = function CompilerSwitches(_, _, b) -> b;;

let _default = CompilerSwitches(false, false, true);;

let parse src =
```

```
  let in_ = IO.input_string src

  in let first_line = input_line in_
  in let second_line = input_line in_

  in close_in in_;
  if second_line <> ""
  then _default, false
  else let hungry = BatString.exists first_line "hungry"
    in let share_stack = BatString.exists first_line "share-stack"
    in let opt_seq = not @@ BatString.exists first_line "optimize-seq"
    in CompilerSwitches(hungry, share_stack, opt_seq),
       if not hungry && not share_stack && opt_seq
       then false
       (* None of the switches appeared in the preamble, in this situation, *)
       (* compiler reads the source from the very beginning. *)
       else true
```

B.14 compiler/traverse.ml

Code Listing

```
(* compiler/traverse.ml -- Author: Zihang Chen (zc2324) *)
open Common;;
open Exc;;

let _ =
  let lexbuf = Lexing.from_channel stdin in
  try
    let ast = Parser.sentence Scanner.token lexbuf in
    let result =
      let traverse x =
        match x with
          Ast.Sentence(words, _) ->
          words_stringify words
      in traverse ast
    in print_endline result
  with
  | LexicalError(s, ln, b) ->
    Printf.printf "(%d,%d) Lexical error: %s.\n" ln b s
  | SyntacticError(s, ln, st, e) ->
    Printf.printf "(%d,%d-%d) Syntactic error: %s.\n"
      ln st e s
```

B.15 compiler/weave.ml

Code Listing

```
(* compiler/weave.ml -- Author: Zihang Chen (zc2324) *)
open Batteries;;
open Exc;;
open Stdint;;
```

```
5
6  let src_file_r = ref "";;
7  let out_file_r = ref "";;
8  let raw_asm_r = ref false;;
9  let raw_text_r = ref false;;
10 let byte_compile_r = ref false;;
11
12 let commands = [
13     ("-o",
14      Arg.Set_string(out_file_r),
15      "Path of the output file");
16
17     ("-r",
18      Arg.Set(raw_asm_r),
19      "Raw asm with labels on");
20
21     ("-t",
22      Arg.Set(raw_text_r),
23      "Raw text asm");
24
25     ("-b",
26      Arg.Set(byte_compile_r),
27      "Byte-compile the given source");
28   ];;
29
30 let () = Arg.parse commands (fun fn -> src_file_r := fn)
31        "The Towel Compiler at your service. Don't panic!";;
32
33 let raw_asm = !raw_asm_r;;
34 let raw_text = !raw_text_r;;
35
36 let src_file = !src_file_r;;
37 let src_inchan = Pervasives.open_in src_file;;
38 let src_content = String.concat "\n" @@ input_list src_inchan;;
39
40 Pervasives.close_in src_inchan;;
41
42 let out_file = !out_file_r;;
43
44 let out = ref "";;
45
46 if !byte_compile_r
47 then let _out =
48   let lexbuf = Lexing.from_string src_content
49   in lexbuf
50      |> Tasm_parser.asm Tasm_scanner.token |> Tasm_bytecode.b_asm
51      |> Bytes.to_string
52   in out := _out
53 else begin
54   let fn_digest = src_content
55                   |> Sha1.string
56                   |> Sha1.to_hex
57                   |> fun x -> String.sub x 0 9
```

```
58
59    in let ir, exp_scope =
60         let in_src, fn = src_content, src_file
61
62         in let sw, has_sw_preamble = Switches.parse(in_src)
63         in let text = if has_sw_preamble
64              then let _, r = String.split src_content "\n"
65                in let _, r = String.split r "\n"
66                in r
67                (* ignore the two-line preamble so that no syntax error
68                   will show up *)
69              else src_content
70
71         in let lexbuf = Lexing.from_string text
72         in try
73
74           let cst = Parser.sentence Scanner.token lexbuf
75           in if (* type checking *) true
76           then Compile.compile cst fn_digest sw
77           else raise TypeError
78
79         with
80         | LexicalError(s, ln, b) ->
81           Printf.printf "(%d,%d) Lexical error: %s.\n" ln b s; exit 0
82         | SyntacticError(s, ln, st, e) ->
83           Printf.printf "(%d,%d-%d) Syntactic error: %s.\n"
84             ln st e s; exit 0
85         | NameNotFoundError(s) ->
86           Printf.printf "Name not found: %s.\n" s; exit 0
87         | TypeError ->
88           Printf.printf "type error\n"; exit 0
89
90    in out := if raw_asm
91      then Tasm_stringify.p_asm ir
92      else if raw_text
93      then ir |> Assemble.assemble |> Tasm_stringify.p_asm
94      else ir |> Assemble.assemble |> Tasm_bytecode.b_asm |> Bytes.to_string;
95    let compose_exp_file x =
96      Hashtbl.fold (fun nstr idx acc
97                    -> acc ^ "\n"
98                       ^ (Printf.sprintf "(%s %su)"
99                            nstr (Uint64.to_string idx)))
100       x ""
101   in let content =
102        Printf.sprintf "\"Automatically generated .e for %s.\"\n\n%s"
103          fn_digest (compose_exp_file exp_scope)
104   in let ochan =
105        if out_file = "-"
106        then Pervasives.stdout
107        else Pervasives.open_out
108          @@ String.concat "."
109            [fst (String.rsplit out_file ".");
110             "e"]
```

```
            (* Replace the t extension with e. *)
  in Pervasives.output_string ochan content; Pervasives.flush ochan;
  if out_file = "-"
  then ()
  else Pervasives.close_out ochan
end;;

let ochan =
     if out_file = "-"
     then Pervasives.stdout
     else Pervasives.open_out out_file
in Pervasives.output_string ochan (!out);
Pervasives.flush ochan;
if out_file = "-"
then () else Pervasives.close_out ochan;;
```

B.16 compiler/wscript

Code Listing

```
# compiler/wscript -- Author: Zihang Chen (zc2324)
#! /usr/bin/env python3

def j(l, ext):
    return ' '.join('%s.%s' % (it, ext) for it in l)

mlj = lambda l: j(l, 'ml')
mlij = lambda l: j(l, 'mli')
cmij = lambda l: j(l, 'cmi')
cmoj = lambda l: j(l, 'cmo')
cmxj = lambda l: j(l, 'cmx')

def _c(n, ext):
    return '%s.%s' % (n, ext)

mln = lambda n: _c(n, 'ml')
mlin = lambda n: _c(n, 'mli')
cmin = lambda n: _c(n, 'cmi')
cmon = lambda n: _c(n, 'cmo')
cmxn = lambda n: _c(n, 'cmx')


def build(ctx):
    mlis = ['ast', 'tasm_ast']
    metalib_mls = ['common']
    lib_mls = ['exc', 'scoping', 'cseg', 'switches', 'config',
               'assemble', 'tasm_stringify', 'tasm_bytecode']
    aff_mlis = ['parser', 'tasm_parser']
    non_exec_mls = ['parser', 'scanner',
                    'tasm_parser', 'tasm_scanner',
                    'exp', 'compile']
    target_mls = ['weave']

```

```
    external_libs = ','.join(ctx.env.LIBS)
    incl_folder = '%s/src/compiler/' % ctx.out_dir
    build_prefix = '%s/' % ctx.out_dir
    # stupid waf!!

    ctx.exec_command([
        'ruby', 'src/tasm/ccg.rb',
        'src/compiler', 'src/tasm/scanner.mll.p'
    ])

    obj_ext = cmxn if ctx.options.compile_natively else cmon
    obj_j = cmxj if ctx.options.compile_natively else cmoj

    oc_rule = '${OCAMLFIND} ${OC} ${DEBUG} -package %s'\
              ' -I %s -c -o ${TGT} ${SRC}'\
              % (external_libs, incl_folder,)

    ctx.add_group('parser')
    ctx(rule='${MENHIR} --base %s/src/compiler/parser ${SRC}' % ctx.out_dir,
        source='parser.mly',
        target='parser.ml parser.mli')
    ctx(rule='${MENHIR} --base %s/src/compiler/tasm_parser ${SRC}'\
        % ctx.out_dir,
        source='tasm_parser.mly',
        target='tasm_parser.ml tasm_parser.mli')

    ctx(rule='${OCAMLLEX} ${SRC} -o %s${TGT}' % build_prefix,
        source='scanner.mll',
        target='scanner.ml')
    ctx(rule='${OCAMLLEX} ${SRC} -o %s${TGT}' % build_prefix,
        source='tasm_scanner.mll',
        target='tasm_scanner.ml')

    ctx(rule=oc_rule,
        source='ast.mli',
        target=ctx.path.get_bld().find_or_declare('ast.cmi'))
    ctx(rule=oc_rule,
        source='tasm_ast.mli',
        target=ctx.path.get_bld().find_or_declare('tasm_ast.cmi'))

    def cpl_oc(n, src_ext_f, tgt_ext_f):
        ctx(rule=oc_rule,
            source=src_ext_f(n),
            target=tgt_ext_f(n))

    ctx.add_group('metalib')
    for it in metalib_mls:
        cpl_oc(it, mln, obj_ext)

    ctx.add_group('lib')
    for it in lib_mls:
        cpl_oc(it, mln, obj_ext)

```

```
    for it in aff_mlis:
        cpl_oc(it, mlin, cmin)

    ctx.add_group('non_exec')
    for it in non_exec_mls:
        cpl_oc(it, mln, obj_ext)

    def cpl_tgt(n):
        ctx(rule=oc_rule,
            source=mln(n),
            target=obj_ext(n))
        ctx(rule='${OCAMLFIND} ${OC} ${DEBUG} -package %s -linkpkg'
                 ' -o %s${TGT} ${SRC}' %\
                 (external_libs, build_prefix),
            source=obj_j(metalib_mls + lib_mls + non_exec_mls + [n]),
            target=n)

    ctx.add_group('tgt')
    for n in target_mls:
        cpl_tgt(n)
```

B.17 vm/amg.rb

Code Listing

```
# vm/amg.rb -- Author: Zihang Chen (zc2324)

typenames = {:fint => 'OVFixedInt',
             :ufint => 'OVUFixedInt',
             :int => 'OVInt',
             :float => 'OVFloat',
             :string => 'OVString',
             :atom => 'OVAtom'}

modnames = {:fint => 'Int64',
            :ufint => 'Uint64',
            :int => 'Big_int',
            :atom => 'Uint64'}

stdint_ops = {'ADD' => 'add', 'SUB' => 'sub', 'MUL' => 'mul',
              'DIV' => 'div', 'MOD' => 'rem', 'AND' => 'logand',
              'OR' => 'logor', 'XOR' => 'logxor', 'NOT' => 'lognot',
              'SHL' => 'shift_left', 'SHR' => 'shift_right',
              'LSHR' => 'shift_right_logical'}

big_int_ops = {'ADD' => 'add_big_int', 'SUB' => 'sub_big_int',
               'MUL' => 'mult_big_int', 'DIV' => 'div_big_int',
               'MOD' => 'mod_big_int', 'AND' => 'and_big_int',
               'OR' => 'or_big_int', 'XOR' => 'xor_big_int',
               'SHL' => 'shift_left_big_int',
               'SHR' => 'shift_right_big_int',
               'LSHR' => 'shift_right_towards_zero_big_int'}

```

```
fint_conv = {:fint => '',
             :ufint => 'Int64.of_uint64',
             :int => 'Int64.of_string @@ Big_int.string_of_big_int',
             :string => 'Int64.of_string',
             :float => 'Int64.of_float',
             :oint => 'Int64.to_int'}

ufint_conv = {:fint => 'Uint64.of_int64',
              :ufint => '',
              :int => 'Uint64.of_string @@ Big_int.string_of_big_int',
              :string => 'Uint64.of_string',
              :float => 'Uint64.of_float',
              :oint => 'Uint64.to_int'}

big_int_conv = {:fint => 'Big_int.big_int_of_string @@ Int64.to_string',
                :ufint => 'Big_int.big_int_of_string @@ Uint64.to_string',
                :int => '', :oint => 'Big_int.int_of_big_int',
                :string => 'Big_int.big_int_of_string'} # no float to big_int function

float_conv = {:fint => 'Int64.to_float',
              :ufint => 'Uint64.to_float',
              :int => 'Big_int.float_of_big_int',
              :string => 'float_of_string',
              :float => ''}

string_conv = {:fint => 'Int64.to_string',
               :ufint => 'Uint64.to_string',
               :int => 'Big_int.string_of_big_int',
               :string => '',
               :float => 'string_of_float'}

gen_conv = lambda do |inst, result_type, conv_hash|
  dts = []

  for k, v in conv_hash
    if k == :oint
      next
    end

    dts.push "#{typenames[k]}(i) -> #{v} i"
  end

  "#{inst} -> (#{typenames[result_type]}(match v with
     #{dts.join("\n    | ")}
   | _ -> failwith \"Invalid type to convert from.\"))"
end

fint_conv_code = gen_conv.call 'TO_FINT', :fint, fint_conv
ufint_conv_code = gen_conv.call 'TO_UFINT', :ufint, ufint_conv
int_conv_code = gen_conv.call 'TO_INT', :int, big_int_conv
float_conv_code = gen_conv.call 'TO_FLOAT', :float, float_conv
string_conv_code = gen_conv.call 'TO_STR', :string, string_conv
convs = [fint_conv_code, ufint_conv_code, int_conv_code,
```

```
          float_conv_code, string_conv_code].join("\n    | ")


conv_def = "let conversions inst dss =
  let v = dspop dss
  in dspush dss (match inst with
    #{convs}
  | _ -> failwith \"Not possible.\");;"

float_ops = {'ADD' => '(+.)', 'SUB' => '(-.)', 'MUL' => '( *.)',
             # avoid (*.) to be part of the comments
             'DIV' => '(/.)'}

op_category = {:fint => stdint_ops, :ufint => stdint_ops, :atom => stdint_ops,
               :int => big_int_ops, :float => float_ops}
conv_category = {:fint => fint_conv, :ufint => ufint_conv, :float => float_conv,
                 :string => string_conv, :int => big_int_conv}

gen_binary_inst = lambda do |inst, supported_data_types, atoms|
  dts = []
  for d in supported_data_types
    opname = op_category[d][inst]
    dot = if modnames.include? d
            '.'
          else
            ''
          end
    dts.push "#{typenames[d]}(i), #{typenames[d]}(j) ->
 #{typenames[d]}(#{modnames[d]}#{dot}#{opname} j i)"
  end

  atom = if atoms
           "    | OVAtom(i), OVAtom(j) -> if not ((is_boolean i) || (is_boolean j))
                 then failwith \"Invalid atom value.\"
                 else OVAtom(Uint64.#{op_category[:ufint][inst]} j i)"
         else
           ""
         end

  "#{inst} -> (match v1, v2 with
      #{dts.join("\n    | ")}
#{atom}
    | _ -> failwith \"Incompatible type to do #{inst}\")"
end

stdtypes = [:fint, :ufint, :int, :float]
inttypes = [:fint, :ufint, :int]
add = gen_binary_inst.call 'ADD', stdtypes, false
sub = gen_binary_inst.call 'SUB', stdtypes, false
mul = gen_binary_inst.call 'MUL', stdtypes, false
div = gen_binary_inst.call 'DIV', stdtypes, false
mod = gen_binary_inst.call 'MOD', inttypes, false
and_ = gen_binary_inst.call 'AND', inttypes, true
```

```
or_ = gen_binary_inst.call 'OR', inttypes, true
xor = gen_binary_inst.call 'XOR', inttypes, false

binary_def = "let binary_arithmetics inst dss =
  let v1 = dspop dss
  in let v2 = dspop dss
  in dspush dss (match inst with
    #{[add, sub, mul, div, mod, and_, or_, xor].join("\n  | ")}
  | _ -> failwith \"Not possible.\");;
"

gen_shift = lambda do |inst, supported_data_types|
  dts = []
  for d in supported_data_types
    opname = op_category[d][inst]
    dot = if modnames.include? d
            '.'
          else
            ''
          end
    dts.push "OVUFixedInt(i), #{typenames[d]}(j) ->
      let ii = Uint64.to_int i in #{typenames[d]}(#{modnames[d]}#{dot}#{opname} j ii)"
  end

  "#{inst} -> (match v1, v2 with
      #{dts.join("\n    | ")}
    | _ -> failwith \"Incompatible type to do #{inst}\")"
end

shl = gen_shift.call 'SHL', inttypes
shr = gen_shift.call 'SHR', inttypes
lshr = gen_shift.call 'LSHR', inttypes

shift_def = "let shift_arithmetics inst dss =
  let v1 = dspop dss
  in let v2 = dspop dss
  in dspush dss (match inst with
    #{[shl, shr, lshr].join("\n | ")}
  | _ -> failwith \"Not possible.\");;
"

gen_unary_instruction = lambda do |inst, supported_data_types, lognot|
  dts = []
  for d in supported_data_types
    if d != :int
      opname = op_category[d][inst]
      dts.push "#{typenames[d]}(i) -> #{typenames[d]}(#{modnames[d]}.#{opname} i)"
    else
      dts.push "OVInt(i) -> OVInt(Big_int.sub_big_int
              (Big_int.minus_big_int i) big_minus_one)"
    end
  end
  if lognot
```

```
    dts.push "OVAtom(i) -> if i = Uint64.one then OVAtom(Uint64.zero)
                      else if i = Uint64.zero then OVAtom(Uint64.one)
                      else failwith \"Invalid atom value.\""
  end

  "#{inst} -> (match v with
      #{dts.join("\n    | ")}
    | _ -> failwith \"Incompatible type to do #{inst}\")"
end

not_ = gen_unary_instruction.call 'NOT', inttypes, true

unary_def = "let unary_arithmetics inst dss =
  let v = dspop dss
  in dspush dss (match inst with
    #{[not_].join("\n | ")}
  | _ -> failwith \"Not possible.\");;
"

open "#{ARGV[0]}/arithmetics.ml", 'w' do |f|
  f.write "open Stdint;;
open T;;
open Nstack;;
open Tasm_ast;;

let is_boolean a = if a = Uint64.one || a = Uint64.zero
  then true
  else false;;

let big_minus_one = Big_int.big_int_of_int (-1);;

"

  f.write conv_def
  f.write binary_def
  f.write shift_def
  f.write unary_def
end
```

B.18 vm/common.ml

Code Listing

```
(* vm/common.ml -- Author: Zihang Chen (zc2324) *)
open Batteries;;
open Stdint;;

(* ========================================
     Counter
   ======================================== *)

let counter = fun () ->
  let cnt = Array.of_list [Uint64.zero]
```

```
11    in fun () -> cnt.(0) <- Uint64.succ cnt.(0); cnt.(0);;
```

B.19 vm/config.ml

Code Listing

```
1  (* vm/config.ml -- Author: Zihang Chen (zc2324) *)
2
3  let libpaths = [Sys.getcwd (); "/Users/wo/ocaml-wksp/Towel/src/towelibs"];;
```

B.20 vm/dscoping.ml

Code Listing

```
1  (* vm/dscoping.ml -- Author: Zihang Chen (zc2324) *)
2  open Nstack;;
3  open T;;
4
5  (* ========================================
6       Scoping routines for Towel virtual machine.
7       Special designed for both dynamic and static flavors.
8
9       We'll use the dynamic one for now though. For static scoping,
10    see sscoping.ml.
11
12      If I were to enable users to add names at runtime, I'll be adding
13    code here.
14    ======================================== *)
15
16 type scope_t = (name_t * module_id_t, vidx_t) Hashtbl.t;;
17
18 let push_scope scp_stk = (Hashtbl.create 512)::scp_stk;;
19
20 let pop_scope = function _::rest -> rest | [] -> [];;
21
22 let npush scp_stk name idx =
23   Hashtbl.replace (List.hd scp_stk) name idx;;
24
25 let npop scp_stk name =
26   Hashtbl.remove (List.hd scp_stk) name;;
27
28 let rec nlookup scp_stk name =
29   match scp_stk with
30     [] -> raise (Exc.NameNotFoundError((fst name), (snd name)))
31   | scp::rest ->
32     try Hashtbl.find scp name
33     with Not_found -> nlookup rest name;;
34
35 let sprint_dscope dscope =
36   Printf.sprintf "{%s}"
37     (String.concat ", "
38        (Hashtbl.fold (fun k v acc -> (Printf.sprintf "%d,%d: (%d,%d)" (fst k) (snd k)
```

```
                          (fst v) (snd v))::acc) dscope []));;

let sprint_dscope_stack scp_stk =
  String.concat " " [string_of_int (List.length scp_stk);
  String.concat "; " (List.map (fun x -> sprint_dscope x) scp_stk)];;
```

B.21 vm/exc.ml

Code Listing

```
(* vm/exc.ml -- Author: Zihang Chen (zc2324) *)
open T;;

exception NameNotFoundError of name_t * module_id_t;;
exception CorruptedScope;;
```

B.22 vm/ext.ml

Code Listing

```
(* vm/ext.ml -- Author: Zihang Chen (zc2324) *)
open Nstack;;
open T;;

module type TowelExtTemplate =
sig
  val extcall: int -> value_t dstack_t dstack_t -> unit
end

let __ext__ = ref None;;
let __get_ext () : (module TowelExtTemplate) =
  match !__ext__ with
    Some(m) -> m
  | None -> failwith "Extension library failed to register itself.";;
```

B.23 vm/imp.ml

Code Listing

```
(* vm/imp.ml -- Author: Zihang Chen (zc2324) *)
open Stdint;;
open Tasm_ast;;
open Dscoping;;

(* ========================================
     Woven file parser
   ======================================== *)

let wvn_filename fn =
  String.concat "." [fn; "w"];;

```

```
let open_woven path =
  Tasm_inv_bytecode.parse_bytecode (Pervasives.open_in path)
```

B.24 vm/jmg.rb

Code Listing

```
# vm/jmg.rb -- Author: Zihang Chen (zc2324)

prelude = "open Tasm_ast;;
open T;;
open Stdint;;
open Vm_t;;

let branch v next_ip =
  function"

assemble_big = lambda {|x| "Big_int.#{x}_big_int i Big_int.zero_big_int"}

match = 'let j_ = Uint64.to_int j in let _ip = match v with'
clause = 'then j_ else next_ip'

numerical_conditionals = [
  ['JNEZ', '<>',
   lambda {'not (Big_int.eq_big_int i Big_int.zero_big_int)'}],

  ['JEZ', '=',
   lambda {assemble_big.call 'eq'}],

  ['JGZ', '>',
   lambda {assemble_big.call 'gt'}],

  ['JGEZ', '>=',
   lambda {assemble_big.call 'ge'}],

  ['JLZ', '<',
   lambda {assemble_big.call 'lt'}],

  ['JLEZ', '<=',
   lambda {assemble_big.call 'le'}],
].map do |xs|
  jn, nb_cond, b_cond = xs

  "| #{jn}(ArgLit(VUFixedInt(j)))
| H#{jn}(ArgLit(VUFixedInt(j))) ->
  #{match}
      OVInt(i) -> if #{b_cond.call}
      #{clause}
    | OVFixedInt(i) -> if Int64.compare i Int64.zero #{nb_cond} 0
      #{clause}
    | OVUFixedInt(i) -> if Uint64.compare i Uint64.zero #{nb_cond} 0
      #{clause}
```

```
    | OVFloat(i) -> if Pervasives.compare i 0.0 #{nb_cond} 0
      #{clause}
    | _ -> failwith \"Unsupported data type.\"
  in _ip"
end
ncs = numerical_conditionals.join "\n\n"

nonnumerical_conditionals = [
  ['JT', 'OVAtom', 'Uint64.zero'],
  ['JF', 'OVAtom', 'Uint64.one'],
].map do |xs|
  jn, cons, cond = xs

  "| #{jn}(ArgLit(VUFixedInt(j)))
| H#{jn}(ArgLit(VUFixedInt(j))) ->
  #{match}
      #{cons}(i) -> if Uint64.compare i #{cond} = 0
      (* Take JF as an example,
         only fall through when i = true, i.e. only jump when i <> true. *)
      then next_ip else j_
    | _ -> j_
  in _ip"
end
nncs = nonnumerical_conditionals.join "\n\n"

conditionals = [ncs, nncs].join "\n\n"

File.open "#{ARGV[0]}/jumps.ml", 'w' do |f|
  f.write prelude
  f.write "\n"
  f.write conditionals
  f.write "\n"
  f.write "| _ -> failwith \"Not possible.\"
"
end
```

B.25 vm/nstack.ml

Code Listing

```
(* vm/nstack.ml -- Author: Zihang Chen (zc2324) *)
open Batteries;;
open T;;
open Printf;;
open Stdint;;

(* ========================================
     Immperative array-based stack.
     The size of it is dynamically allocated.
   ======================================== *)


(* Only for debugging purposes. *)
```

```
let rec string_of_value v =
  match v with
    OVInt(i) -> Big_int.string_of_big_int i
  | OVFixedInt(i) -> Int64.to_string i
  | OVUFixedInt(i) -> Uint64.to_string i
  | OVFloat(f) -> string_of_float f
  | OVAtom(a) -> Uint64.to_string a
  | OVString(s) -> s
  | OVList(rs) ->
    sprintf "[%s]"
      (String.concat " " (List.map string_of_value !rs))
  | OVTuple(rs) ->
    sprintf "[@ %s]"
      (String.concat " " (List.map string_of_value !rs))
  | OVFunction(f) ->
    sprintf "**%s: %d,%d,<closure set>"
      (if f.is_partial then "partial-fun" else "fun")
      f.st f.mod_id
  | _ -> "**abstract value";;

type 'a dstack_t = 'a BatDynArray.t;;
exception PhonyEmptyStack;;

let dinit () = BatDynArray.make 128;;

let dsp ds = BatDynArray.length ds - 1;;

let dpush ds v =
  BatDynArray.add ds v;;

let _safe_dsp ds = pred (dsp ds);;

let dtop ds =
  let top = BatDynArray.last ds
  in if top = OVPhony
  then raise PhonyEmptyStack
  else top;;

let dis_empty ds = if BatDynArray.length ds = 0
    then 1
    else if BatDynArray.last ds = OVPhony
    then 2
    else 0;;

let dswap ds idx1 idx2 =
  let t = BatDynArray.get ds idx1
  in BatDynArray.set ds idx1 (BatDynArray.get ds idx2);
  BatDynArray.set ds idx2 t;;

let dpop ds =
  let popped = dtop ds
  in begin
    BatDynArray.delete_last ds;
```

```
    if popped = OVPhony
    then raise PhonyEmptyStack
    else popped
  end;;

let dpurge = BatDynArray.clear;;

let dspush dss v =
  let stack = BatDynArray.last dss
  in dpush stack v;;

let dstop dss =
  let stack = BatDynArray.last dss
  in dtop stack;;

let dspop dss =
  let stack = BatDynArray.last dss
  in dpop stack;;

let dsis_empty dss =
  let stack = BatDynArray.last dss
  in dis_empty stack;;

let dspurge (* purge the top stack *) dss =
  let stack = BatDynArray.last dss
  in begin
    BatDynArray.delete_last dss;
    dpurge stack
  end;; (* Maybe an overkill? Maybe not. *)

let sprint_ds ds = Printf.sprintf "[|%s|]"
    (String.concat ", "
       (BatDynArray.fold_left
          (fun acc x -> (string_of_value x)::acc)
          [] ds));;

let sprint_dss dss = Printf.sprintf "[[| %s |]]"
    (String.concat " | "
       (BatDynArray.fold_left
          (fun acc x -> (sprint_ds x)::acc)
          [] dss));;

(* ========================================
     Below are some indexing functions.
   ======================================== *)

type vidx_t = int * int;; (* ds idx, dss idx *)

let dval dss idx =
  let ds = BatDynArray.get dss (snd idx)
  in BatDynArray.get ds (fst idx);;

let snd_ds dss = BatDynArray.get dss ((_safe_dsp dss));;
```

B.26 vm/nvm.ml

Code Listing

```
(* vm/nvm.ml -- Author: Zihang Chen (zc2324) *)
open Batteries;;
open T;;
open Stdint;;
open Tasm_ast;;
open Imp;;
open Dscoping;; (* Change to Sscoping for static scoping. *)
open Jumps;;
open Config;;
open Vm_t;;
open Nstack;;
open Printf;;
open Common;;
open Arithmetics;;
open Ext;;

let vm_name = Uint64.to_int;;
let vm_mod_id = Uint64.to_int;;
let to_pc = Uint64.to_int;;

let modules:module_table_t = Hashtbl.create 64;;

let opened_exts = Hashtbl.create 64;;

let name_id_tick_gen () =
  let table = Hashtbl.create 512
  in fun name ->
    try Hashtbl.find table name
    with Not_found -> Hashtbl.replace table name name; name;;

let _MAIN_MODULE_ID = 1;;
let _SELF_MODULE_ID = 0;;
let module_id_tick = let c = Common.counter ()
  in fun () -> vm_mod_id (c ());;
ignore (module_id_tick ());; (* tick for _MAIN_MODULE_ID *)

let ext_id_tick_gen () =
  let table = Hashtbl.create 512
  in let ext_id_tick =
       let c = Common.counter ()
       in fun () -> vm_mod_id (c ())
  in fun fn ->
    try Hashtbl.find table fn
    with Not_found -> let id = ext_id_tick ()
      in Hashtbl.replace table fn id; id;;

let ext_id_tick = ext_id_tick_gen ();;

let tos = List.hd;;
```

```
let ntos = List.tl;;

let ctxs_ = [{mod_id = _MAIN_MODULE_ID; ret_addr = -1;
              list_make_stack = [];
              is_tail_recursive_call = false;
              scp_count = 0;
              curfun = {st = -1;
                        mod_id = _MAIN_MODULE_ID;
                        closure = Hashtbl.create 1;
                        is_partial = false}}];;

let show_ctx c =
  fprintf stderr "{ctx: %d,%d}" c.mod_id c.ret_addr;;

let rec show_ctxs = function
    ctx::rest -> (show_ctx ctx; fprintf stderr " "; show_ctxs rest)
  | [] -> fprintf stderr "\n";;

let rec find_module mod_str = function
    path::rest ->
    let possible_mod_path = Filename.concat path (mod_str ^ ".w")
    in if BatSys.file_exists possible_mod_path
    then open_woven possible_mod_path
    else find_module mod_str rest
  | [] -> failwith (sprintf "Requested module %s not found." mod_str);;

let rec find_ext ext_str = function
    path::rest ->
    let possible_ext_path = Filename.concat path ext_str
    in if BatSys.file_exists possible_ext_path
    then possible_ext_path
    else find_ext ext_str rest
  | [] -> failwith (sprintf "Requested extension %s not found." ext_str);;

let rec string_of_value v =
  match v with
    OVInt(i) -> Big_int.string_of_big_int i
  | OVFixedInt(i) -> Int64.to_string i
  | OVUFixedInt(i) -> Uint64.to_string i
  | OVFloat(f) -> string_of_float f
  | OVAtom(a) ->
    if Uint64.compare a Uint64.one = 0
    then "true"
    else if Uint64.compare a Uint64.zero = 0
    then "false"
    else sprintf "%sa" (Uint64.to_string a)
  | OVString(s) -> s
  | OVList(rs) ->
    sprintf "[%s]"
      (String.concat " " (List.map string_of_value !rs))
  | OVTuple(rs) ->
    sprintf "[@ %s]"
      (String.concat " " (List.map string_of_value !rs))
```

```
103     | OVFunction(f) ->
104       sprintf "**%s: %d,%d,<closure set>"
105         (if f.is_partial then "partial-fun" else "fun")
106         f.st f.mod_id
107     | OVTypeHint(t) ->
108       "**type value"
109     | OVPhony ->
110       "$$"
111     | _ -> "**abstract value";;
112
113   let exec should_trace should_warn insts =
114     let rec __exec ctxs flags ip =
115       let next_ip = succ ip
116       in let line = flags.curmod.insts.(ip)
117
118       in let trace msg =
119            if should_trace
120            then begin
121              show_ctxs ctxs;
122              fprintf stderr "**(%d,%d) -- %s\n"
123                flags.curmod.id ip msg
124            end else ()
125
126       (* in let trace_u64 u = *)
127       (*       if should_trace *)
128       (*       then fprintf stderr "%s" (Uint64.to_string u) *)
129       (*       else () *)
130
131       in let tvm_warning msg =
132            if should_warn
133            then fprintf stderr "xx(%d,%d) -- %s\n"
134                flags.curmod.id ip msg
135            else ()
136
137       in let curmod = flags.curmod
138
139       in let dss = flags.dss
140       in let scps = flags.scps
141
142       in let list_make_stack = (tos ctxs).list_make_stack
143
144       in let push_cur_ip cur_fun =
145            let new_closure = Hashtbl.copy cur_fun.closure
146            in Hashtbl.iter
147                (fun k v -> if Hashtbl.mem cur_fun.closure k
148                  then ()
149                  else Hashtbl.replace new_closure k v)
150                (* Copy all the values bound in the last context's function's
151                   closure in this function's closure (of course a copy) so that
152                   multiple level of binding can work. *)
153                (* This is potentially memory-ineffective. But without this,
154                   quicksort (with bind) won't work, although I believe it
155                   should work with the presence of install, despite which only
```

```
156                takes care for scopes created by functions. *)
157             (tos ctxs).curfun.closure;
158          {mod_id = curmod.id;
159           is_tail_recursive_call = false;
160           ret_addr = next_ip;
161           scp_count = 0;
162           list_make_stack = [];
163           curfun = {cur_fun with closure = new_closure}}::ctxs
164
165      in let __put_val x = if List.length list_make_stack = 0
166           then dspush dss x
167           else let cur_ref = tos list_make_stack
168             in cur_ref := x::(!cur_ref)
169
170      in let return _ nscps =
171        let tctx = tos ctxs
172        in let tmod = Hashtbl.find modules tctx.mod_id
173        in begin
174           (* Wipe out current stack for GC. *)
175          if dsis_empty dss = 0
176          then let r = dspop dss
177            in begin
178              dspurge dss;
179              let __ret_lmstk = (ctxs |> ntos |> tos).list_make_stack
180                  (* second top ctx *)
181              in if List.length __ret_lmstk = 0
182              then dspush dss r
183              else let cur_ref = tos __ret_lmstk
184                in cur_ref := r::(!cur_ref)
185                (* Copy return value to caller's dss. *)
186            end else dspurge dss;
187          (* If we purge (and pop) the current stack then copy the return value
188             there won't be a problem if the two contexts are in the same
189             module. *)
190          __exec (ntos ctxs) {flags with curmod = tmod;
191                                         scps = nscps} tctx.ret_addr
192        end
193
194      in let inc_scp_count () =
195           {(tos ctxs) with scp_count = succ (tos ctxs).scp_count}::
196           (ntos ctxs)
197
198      in let invoke_regular fr =
199           let nmod = Hashtbl.find modules fr.T.mod_id
200           in __exec (push_cur_ip fr)
201             {flags with curmod = nmod}
202             fr.st
203
204      (* `dsp' points to the next slot of TOS. *)
205      in let tos_idx () = ((dsp (BatDynArray.last dss)),
206                           (dsp dss))
207
208      in let __push_new_ref_to_toctxs nr =
```

```
209           {(tos ctxs) with list_make_stack = nr::list_make_stack}
210           ::(ntos ctxs)
211
212       in let push_new_list type_ =
213           (* 1 for list, 2 for tuple *)
214           let nr = ref []
215           in let nv = if type_ = 1 then OVList(nr) else OVTuple(nr)
216           in if List.length list_make_stack = 0
217           then begin
218             dspush dss nv;
219             __exec (__push_new_ref_to_toctxs nr) flags next_ip
220           end else let cur_ref = tos list_make_stack
221             in let () = cur_ref := nv::(!cur_ref)
222             in (* No push operation in this situation. *)
223             __exec (__push_new_ref_to_toctxs nr) flags next_ip
224
225       in let end_list () =
226           let _tos = tos list_make_stack
227           in _tos := List.rev !_tos;
228           __exec ({(tos ctxs) with list_make_stack = ntos list_make_stack}
229                   ::(ntos ctxs))
230             flags
231             next_ip
232
233       in let resolve_name n m =
234           let m = Hashtbl.find curmod.imports m
235           in if m <> curmod.id
236           then Hashtbl.find (Hashtbl.find modules m).exs n
237               (* This module is destined to exist.
238                  1. You have to import the module's .e file to be
239                  able to compile code that actually uses values in
240                  this module.
241                  2. Therefore when you resolving names in this module,
242                  this module is destined to be imported already.
243                  3. Therefore you can't miss this name here.
244                  4. So no need for save values other than current module
245                  to closure. *)
246
247           else try let rn, rm = nlookup scps (n, m)
248               in if rm < 0
249               then (* It's in closure. *)
250                 !(Hashtbl.find (tos ctxs).curfun.closure (rn, -rm))
251               else dval dss (rn, rm)
252             (* Check local scope first, because names captured in closure
253                might be shadowed in current scope. *)
254             with Exc.NameNotFoundError(_) -> failwith "name not found"
255                | Not_found -> failwith "external name not found in tctx.curfun.closure"
256
257       in let inst =
258           match line with
259             Line(_, i) -> i
260           | CLine(_, Some(i)) ->
261             tvm_warning "found a CLine, maybe a Cseg bug?"; i
```

```
262             | CLine(_, None) ->
263               tvm_warning "found an empty CLine, definitely a bug."; IDLE
264
265       in let put_val x = __put_val x;
266            __exec ctxs flags next_ip
267
268       in trace (sprint_dss dss);
269       match inst with
270         PUSH_LIT(ArgLit(lit)) -> trace "pushing lit";
271         let nv = match lit with
272             VInt(i) -> OVInt(i)
273           | VFloat(f) -> OVFloat(f)
274           | VString(s) -> OVString(s)
275           | VUFixedInt(uf) -> OVUFixedInt(uf)
276           | VFixedInt(f) -> OVFixedInt(f)
277           | VAtom(a) -> OVAtom(a)
278         in put_val nv
279
280       | PUSH_LNIL -> trace "pushing lnil";
281         push_new_list 1
282
283       | PUSH_TNIL -> trace "pushing tnil";
284         push_new_list 2
285
286       | END_LIST -> trace "ending list";
287         end_list ()
288
289       | END_TUPLE -> trace "ending tuple";
290         end_list ()
291
292       | LOAD_EXT -> trace "loading ext";
293         let name =  (match dspop dss with
294               OVString(s) -> s
295             | _ -> failwith "incompatible type of argument to load-ext")
296         in let fn = find_ext (Dynlink.adapt_filename name) libpaths
297         in (* Maybe need to query the table whether this ext has been loaded. *)
298         (try Dynlink.loadfile fn
299          with Dynlink.Error(s) ->
300          match s with
301            Dynlink.Not_a_bytecode_file(a) -> failwith (sprintf "1%s\n" a)
302          | Dynlink.Inconsistent_import(a) -> failwith (sprintf "2%s\n" a)
303          | Dynlink.Unavailable_unit(a) -> failwith (sprintf "3%s\n" a)
304          | Dynlink.Unsafe_file -> failwith (sprintf "4%s\n" "unsafe file")
305          | Dynlink.Linking_error(a, b) ->
306            (match b with
307               Dynlink.Undefined_global(c)
308               -> failwith (sprintf "5%s\n" c)
309             | Dynlink.Unavailable_primitive(c)
310               -> failwith (sprintf "5%s\n" c)
311             | Dynlink.Uninitialized_global(c)
312               -> failwith (sprintf "5%s\n" c))
313          | Dynlink.Corrupted_interface(a) -> failwith (sprintf "6%s\n" a)
314          | Dynlink.File_not_found(a) -> failwith (sprintf "7%s\n" a)
```

```
315         | Dynlink.Cannot_open_dll(a) -> failwith (sprintf "8%s\n" a)
316         | Dynlink.Inconsistent_implementation(a) -> failwith (sprintf "9%s\n" a));
317
318       let uid = ext_id_tick fn
319       in let module ThisModule = (val __get_ext () : TowelExtTemplate)
320       in Hashtbl.replace opened_exts uid
321         (module ThisModule : TowelExtTemplate);
322       dspush dss (OVUFixedInt(Uint64.of_int uid));
323       __exec ctxs flags next_ip
324
325     | EXTCALL -> trace "loading ext";
326       let ext_id = match (dspop dss) with
327           OVUFixedInt(i) -> Uint64.to_int i
328         | _ -> failwith "incompatible type of argument 1 to extcall"
329       in let cn = match (dspop dss) with
330             OVUFixedInt(i) -> Uint64.to_int i
331           | _ -> failwith "incompatible type of argument 2 to extcall"
332       in let module E =
333            (val Hashtbl.find opened_exts ext_id : TowelExtTemplate)
334       in E.extcall cn dss;
335       __exec ctxs flags next_ip
336
337     | INSTALL -> trace "installing closure";
338       (* This instruction effectively notifies the scoping mechanism about
339          all the captured values that are in closure so that when resolving
340          a name, we don't have to explicitly check the closure. *)
341       let fr = (tos ctxs).curfun
342       in Hashtbl.iter
343         (fun k _ -> let n, m = k
344          in npush scps k (n, -m)) fr.closure;
345       __exec ctxs flags next_ip
346
347     | SWEEP -> trace "sweeping for tail recursive stacks";
348       if (tos ctxs).is_tail_recursive_call
349       then begin
350         dspurge dss;
351         BatDynArray.add dss (dinit ())
352       end else ();
353       __exec ctxs flags next_ip
354
355     | PACK -> trace "packing";
356       let n = match (dspop dss) with
357           OVUFixedInt(i) -> Uint64.to_int i
358         | OVFixedInt(i) -> let r = Int64.to_int i
359           in if r < -1
360           then failwith "Invalid argument to pack (arg < -1)."
361           else r
362         | OVInt(i) -> let r = Big_int.int_of_big_int i
363           in if r < -1
364           then failwith "Invalid argument to pack (arg < -1)."
365           else r
366         | _ -> failwith "Invalid type of argument to pack."
367       in if n = -1
```

```
368         then let rec __pop_until_empty acc =
369                let empty = dsis_empty dss
370                in if empty = 0
371                then __pop_until_empty ((dspop dss)::acc)
372                else if empty = 1
373                then acc
374                else begin let ds = BatDynArray.last dss
375                  in BatDynArray.delete_last ds;
376                  acc
377                end
378           in let acc = __pop_until_empty []
379           in dspush dss (OVList(ref acc));
380                __exec ctxs flags next_ip
381         else let nl = List.fold_left (fun acc _ ->
382             (dspop dss)::acc) [] (BatList.range 1 `To n)
383           in dspush dss (OVList(ref nl));
384           __exec ctxs flags next_ip
385
386       | UNPACK -> trace "unpacking";
387         let ns = match (dspop dss) with
388             OVList(rls) -> !rls
389           | OVTuple(rls) -> !rls
390           | _ -> failwith "Invalid type of argument to unpack."
391         in List.iter (fun x -> dspush dss x) ns; (* TODO put_val or dspush? *)
392         __exec ctxs flags next_ip
393
394       | TYPE -> trace "typing";
395         let n = match (dspop dss) with
396             OVInt(_) -> OVTypeHint(THInt)
397           | OVAtom(_) -> OVTypeHint(THAtom)
398           | OVFixedInt(_) -> OVTypeHint(THFixedInt)
399           | OVUFixedInt(_) -> OVTypeHint(THUFixedInt)
400           | OVString(_) -> OVTypeHint(THString)
401           | OVFloat(_) -> OVTypeHint(THFloat)
402           | OVList(_) -> OVTypeHint(THList)
403           | OVPhony -> OVTypeHint(THPhony)
404           (* don't think you can do this *)
405           | OVFunction(_) -> OVTypeHint(THFunction)
406           | OVTuple(_) -> OVTypeHint(THTuple)
407           | OVNil -> OVTypeHint(THNil)
408           | OVTypeHint(_) -> OVTypeHint(THType)
409         in dspush dss n;
410         __exec ctxs flags next_ip
411
412       | DUP -> trace "duplicating";
413         let v = dstop dss
414         in put_val v
415
416       | READ -> trace "reading";
417         let n = Pervasives.input_line Pervasives.stdin
418         in dspush dss (OVString(n));
419         __exec ctxs flags next_ip
420
```

```
421       | CAR -> trace "pushing list head";
422         let tos = dspop dss
423         in (match tos with
424               OVList(rls) -> dspush dss (List.hd !rls)
425             | OVString(s) -> dspush dss (OVString(BatString.head s 1))
426             | _ -> failwith "Non hd-able value type.");
427         __exec ctxs flags next_ip
428
429       | CDR -> trace "pushing list tail";
430         let tos = dspop dss
431         in (match tos with
432               OVList(rls) -> dspush dss (OVList(ref (List.tl !rls)))
433             | OVString(s) -> dspush dss (OVString(BatString.tail s 1))
434             | _ -> failwith "Non tl-able value type.");
435         __exec ctxs flags next_ip
436
437       | CONS -> trace "consing list";
438         let l = dspop dss
439         in let elem = dspop dss
440         in (match l with
441               OVList(rls) -> dspush dss (OVList(ref (elem::(!rls))))
442             | OVString(s) -> let ss = match elem with
443                   OVString(ss) -> ss
444                 | _ -> failwith "Cons'ing a non-string to a string."
445               in dspush dss (OVString(String.concat "" [ss; s]))
446             | _ -> failwith "Cons'ing a non-list value.");
447         __exec ctxs flags next_ip
448
449       | LIST_EMPTY -> trace "pushing is_empty_list";
450         let tos = dspop dss
451         in let judge rls = if List.length !rls = 0
452              then OVAtom(Uint64.one)
453              else OVAtom(Uint64.zero)
454         in (match tos with
455               OVList(rls)
456             | OVTuple(rls) -> dspush dss (judge rls)
457             | OVString(s) -> if String.length s = 0
458               then dspush dss (OVAtom(Uint64.one))
459               else dspush dss (OVAtom(Uint64.zero))
460             | _ -> failwith "Non is_empty-able value type.");
461         __exec ctxs flags next_ip
462
463       | TUPLE_AT -> trace "getting tuple element";
464         let n = match dspop dss with
465             OVUFixedInt(i) -> Uint64.to_int i
466           | OVFixedInt(i) -> let x = Int64.to_int i
467             in if x < 0
468             then failwith "Invalid index for a tuple."
469             else x
470           | OVInt(i) -> let x = Big_int.int_of_big_int i
471             in if x < 0
472             then failwith "Invalid index for a tuple."
473             else x
```

```
474           | _ -> failwith "Invalid index type for a tuple."
475         in let tuple = match dspop dss with
476               OVTuple(rls) -> !rls
477             | _ -> failwith "Invalid target for tuple-at."
478         in dspush dss (BatList.at tuple n);
479         __exec ctxs flags next_ip
480
481       | POP -> trace "popping";
482         (try
483            ignore (dspop dss);
484            __exec ctxs flags next_ip
485          with PhonyEmptyStack -> failwith "popping from empty stack; phony.")
486
487       | PUSH_FUN(ArgLit(VUFixedInt(st))) -> trace "making function";
488         let nf = OVFunction({
489             st = to_pc st;
490             mod_id = flags.curmod.id;
491             closure = Hashtbl.create 32;
492             is_partial = false;
493           })
494         in put_val nf
495
496       | PUSH_NAME(ArgLit(VUFixedInt(_nid)), ArgLit(VUFixedInt(_mid))) ->
497         trace "pushing name";
498         let nid = vm_name _nid
499         in let mid = vm_mod_id _mid
500         in let v = resolve_name nid mid
501         in put_val v
502
503       | CALL(ArgLit(VUFixedInt(st))) -> trace "pushing function";
504         let nfrec = {st = to_pc st;
505                      mod_id = flags.curmod.id;
506                      closure = Hashtbl.create 32;
507                      is_partial = false}
508         in
509         __exec (push_cur_ip nfrec) flags (to_pc st)
510
511       | INVOKE -> trace "invoking tos";
512         let f = dspop dss
513         in (match f with
514               OVFunction(frec) -> invoke_regular frec
515             | _ -> failwith "invoking non-function value")
516
517       | ADD
518       | SUB
519       | MUL
520       | DIV
521       | MOD
522       | AND
523       | OR
524       | XOR -> binary_arithmetics inst dss;
525         __exec ctxs flags next_ip
526
```

```
527       | TO_FINT
528       | TO_UFINT
529       | TO_INT
530       | TO_FLOAT
531       | TO_STR -> conversions inst dss;
532         __exec ctxs flags next_ip
533
534       | SHL
535       | SHR
536       | LSHR ->
537         shift_arithmetics inst dss;
538         __exec ctxs flags next_ip
539
540       | NOT ->
541         unary_arithmetics inst dss;
542         __exec ctxs flags next_ip
543
544       | EQU -> trace "testing equality";
545         let v1 = dspop dss
546         in let v2 = dspop dss
547         in let tf = (match v1, v2 with
548             (* [| v2 | v1 |], when evaluating v2 v1 -, we want v2 - v1. *)
549               OVFixedInt(i), OVFixedInt(j) ->
550               Int64.compare i j
551             | OVUFixedInt(i), OVUFixedInt(j) ->
552               Uint64.compare j i
553             | OVFloat(i), OVFloat(j) ->
554               tvm_warning "testing equality between float numbers.";
555               Pervasives.compare j i
556             | OVInt(i), OVInt(j) ->
557               if Big_int.eq_big_int j i then 0 else 1
558             | OVAtom(i), OVAtom(j) ->
559               Uint64.compare i j
560             | OVString(i), OVString(j) ->
561               Pervasives.compare i j
562             | OVTypeHint(i), OVTypeHint(j) ->
563               Pervasives.compare i j
564             | _ -> 1 (* Equality of non-equal values are 1. *))
565         in dspush dss (if tf = 0
566                        then OVAtom(Uint64.one)
567                        else OVAtom(Uint64.zero));
568           __exec ctxs flags next_ip
569
570       | JUMP(ArgLit(VUFixedInt(p))) -> trace "jumping";
571         __exec ctxs flags (to_pc p)
572
573       | JE(ArgLit(VUFixedInt(p))) -> trace "je";
574         let j = let r = dsis_empty dss
575           in if r = 1
576           then (to_pc p)
577           else if r = 2 (* Phony empty stack. *)
578           then begin
579             let ds = BatDynArray.last dss
```

```
580             in BatDynArray.delete_last ds;
581             (to_pc p)
582           end else next_ip
583         in __exec ctxs flags j
584
585       | HJE(ArgLit(VUFixedInt(p))) -> trace "hje";
586         (* Really cannot do much about this duplication. *)
587         let j = let r = dsis_empty dss
588           in if r = 1
589           then (to_pc p)
590           else if r = 2
591           then begin
592             let ds = BatDynArray.last dss
593             in BatDynArray.delete_last ds; (* Pop the OVPhony. *)
594             (to_pc p)
595           end else next_ip
596         in __exec ctxs flags j
597
598       | HJNE(ArgLit(VUFixedInt(p)))
599       | JNE(ArgLit(VUFixedInt(p))) -> trace "jne";
600         let j = if dsis_empty dss = 0
601           then (to_pc p)
602           else next_ip
603         in __exec ctxs flags j
604
605       | JT(_)
606       | JF(_)
607       | JEZ(_)
608       | JGZ(_)
609       | JLZ(_)
610       | JNEZ(_)
611       | JGEZ(_)
612       | JLEZ(_) ->
613         __exec ctxs flags
614           (branch (dstop dss) next_ip inst)
615
616       | HJT(_)
617       | HJF(_)
618       | HJEZ(_)
619       | HJGZ(_)
620       | HJLZ(_)
621       | HJNEZ(_)
622       | HJGEZ(_)
623       | HJLEZ(_) ->
624         __exec ctxs flags
625           (branch (dspop dss) next_ip inst)
626
627       | SHARED_RET -> trace "shared returning";
628         let tctx = tos ctxs
629         in __exec (ntos ctxs) {flags with
630                                curmod = Hashtbl.find modules tctx.mod_id}
631           tctx.ret_addr
632
```

```
633       | RET -> trace "returning";
634         return () scps
635
636       | PUSH_PHONY -> trace "pushing phony";
637         put_val OVPhony; (* push-phony is a low level instruction, if we were
638                             to push a phony into string, use put_val here. *)
639         __exec ctxs flags next_ip
640
641       | CLOSURE(ArgLit(VUFixedInt(_nid))) ->
642         trace "adding to closure";
643         let nid = vm_name _nid
644         in let mid = 0
645         (* Because resolve_name will absolutify mid for us, so put 0 here. *)
646         in (match dstop dss with
647               OVFunction(f) ->
648               let v = resolve_name nid mid
649               in Hashtbl.replace f.closure (nid,
650                                             Hashtbl.find curmod.imports mid)
651                 (ref v)
652             | _ -> failwith "Adding captured value to non-function.
653   Something is wrong with the compiler.");
654         __exec ctxs flags next_ip
655
656       | FUN_ARG(ArgLit(VUFixedInt(_nid))) -> trace "fun argumenting";
657         let nid = vm_name _nid
658         in let nid = curmod.name_id_tick nid
659         in let _f = (tos ctxs).curfun
660         in let f = {_f with closure = Hashtbl.copy _f.closure}
661         (* Because we are storing function arguments in closure, we must copy
662            the function every time we execute a fun-arg, just to make sure
663            recursive functions work. For example, if we are recursively calling
664            our function with new arguments A = 10, B = 11; if we are not copying
665            the function closure table, {ctx5}'s curfun.closure.A would be 10,
666            thus overwriting ctx5's execution stack.
667
668            ({ctx5 curfun = {closure = {A = 1; B = 2}}})
669            ({ctx4 curfun = {closure = {A = 3; B = 6}}})
670            ({ctx3 curfun = {closure = {A = 3; B = 1}}})
671            ...
672
673            I know copying is slow, but with this fun-arg approach, I have no
674            choice. *)
675         in let remove_phony_if_any ds =
676              let r = dis_empty ds
677              in if r = 1
678              then () (* It's really empty. *)
679              else if r = 2
680              then BatDynArray.delete_last ds
681
682         in let steal_arg () =
683              if (tos ctxs).is_tail_recursive_call
684              then if dsis_empty dss <> 0
685                then begin
```

```
            if dsis_empty dss = 2
            (* Definitely want to remove this phony, or the code after
               this function won't work. *)
            then remove_phony_if_any (BatDynArray.last dss)
            (* The difference with non tail recursive functions is that
               we steal arguments from different stacks. *)
            else ();
            true, OVNil
          end else false, dspop dss
        else let second_level_stack = snd_ds dss
          in if dis_empty second_level_stack <> 0
          then begin
            if dis_empty second_level_stack = 2
            then remove_phony_if_any second_level_stack
            else ();
            true, OVNil
          end else
            false, dpop second_level_stack

      in let no_more_argument, stolen_arg =
           if f.is_partial
           then if Hashtbl.mem f.closure (nid, curmod.id)
             then false, !(Hashtbl.find f.closure (nid, curmod.id))
             (* We found what we want in the closure set. Just get the
                argument from it.

                This must be checked first to ensure that partial applied
                functions work as expected. *)

             (* Otherwise, we steal argument from others. *)
             else steal_arg ()
           else steal_arg ()
      in if no_more_argument
      then begin
        let new_closure = Hashtbl.copy f.closure
        in let nf = OVFunction({f with closure = new_closure;
                                       is_partial = true})
        in dspush dss nf; (* Actually there is no point in changing the dspush
                             here. Because you cannot create a list before the
                             last fun-arg is executed. *)
        return () (ntos scps) (* This is where I found the scope leaking.
                                 Basically with partial functions' creation. *)
      end else begin
        Hashtbl.replace f.closure (nid, curmod.id) (ref stolen_arg);
        (* Store the function argument here at f.closure. *)
        npush scps (nid, curmod.id) (nid, -curmod.id);
        (* Make the function argument visible to inner scopes so that the
           following code will work.
           fun A,
             fun B,
               A *)
        __exec ({(tos ctxs) with curfun = f}::(ntos ctxs))
          (* Modify the toctx with the newly modified version of the
```

```
739                function. *)
740              flags next_ip
741          end
742
743        | REVERSE -> trace "reversing";
744          let n = Uint64.to_int (match dspop dss with
745                OVUFixedInt(u) -> u
746              | _ -> failwith "Non-compatible type for reverse.")
747          in let end_ = (BatDynArray.length (BatDynArray.last dss)) - 1
748          in let st = if end_ - n + 1 >= 0 then end_ - n + 1 else 0
749          in let rec _swap_them_all idx1 idx2 =
750               if idx1 < idx2
751               then begin
752                 dswap (BatDynArray.last dss) idx1 idx2;
753                 _swap_them_all (succ idx1) (pred idx2)
754               end else ()
755          in _swap_them_all st end_;
756          __exec ctxs flags next_ip
757
758        | EVAL_TAIL(ArgLit(VUFixedInt(_nid)), ArgLit(VUFixedInt(_mid))) ->
759          trace "tail recursive call";
760          let nid = vm_name _nid
761          in let mid = vm_mod_id _mid
762          in let v = resolve_name nid mid
763
764          in let rec remove_scps _scps count =
765               if count = 0
766               then _scps
767               else remove_scps (pop_scope _scps) (pred count)
768
769          in (match v with
770                OVFunction(frec) ->
771                let new_ctxs = {(tos ctxs) with curfun = frec;
772                                                scp_count = 0;
773                                                is_tail_recursive_call = true;
774                                                list_make_stack = []}
775                               ::(ntos ctxs)
776                in let tail_ip = frec.st |> succ
777                (* Bypass the push-scope instruction. *)
778                in if frec.mod_id = curmod.id
779                then __exec new_ctxs
780                    {flags with scps = remove_scps scps (tos ctxs).scp_count}
781                    tail_ip
782                else __exec new_ctxs
783                    {flags with scps = remove_scps scps (tos ctxs).scp_count;
784                                curmod = Hashtbl.find modules frec.mod_id}
785                    tail_ip
786              | _ -> failwith "Tail recursing a non-function.")
787
788          | EVAL_AND_PUSH(ArgLit(VUFixedInt(_nid)), ArgLit(VUFixedInt(_mid))) ->
789            trace "evaluating name";
790            let nid = vm_name _nid
791            in let mid = vm_mod_id _mid
```

```
792           in let v = resolve_name nid mid
793           in (match v with
794                 OVFunction(frec) -> invoke_regular frec
795
796               | OVInt(_)
797               | OVAtom(_)
798               | OVFixedInt(_)
799               | OVUFixedInt(_)
800               | OVString(_)
801               | OVFloat(_)
802               | OVList(_)
803               | OVTuple(_)
804               | OVNil
805               | OVTypeHint(_) ->
806                 put_val v
807               | _ ->
808                 __exec ctxs flags next_ip)
809
810         | IMPORT(ArgLit(VUFixedInt(uid))) ->
811           trace "importing";
812           let mod_str = match dspop dss with
813               OVString(s) -> s
814             | _ -> failwith "invalid type for a mod_str, may be a compiler bug"
815           in (try let _ = Hashtbl.find curmod.imports (vm_mod_id uid)
816                 in __exec ctxs flags next_ip (* Do nothing since we have already
817                                                 imported the module before. *)
818               with Not_found ->
819                 let w_insts = find_module mod_str libpaths
820                 in let abs_uid = module_id_tick ()
821
822                 in let module_ = {id = abs_uid;
823                                   insts = w_insts;
824                                   imports = Hashtbl.create 512;
825                                   name_id_tick = name_id_tick_gen ();
826                                   exs = Hashtbl.create 512}
827
828                 in (Hashtbl.replace modules abs_uid module_);
829                 (Hashtbl.replace curmod.imports (vm_mod_id uid) abs_uid);
830                 (Hashtbl.replace module_.imports _SELF_MODULE_ID abs_uid);
831                 (* Update the imports proxy for mapping from rel to abs. *)
832                 (__exec
833                    (push_cur_ip {st = 0; mod_id = curmod.id;
834                                  closure = Hashtbl.create 1;
835                                  is_partial = false})
836                    {flags with curmod = module_}
837                    0))
838
839         | BIND(ArgLit(VUFixedInt(uid))) -> trace "binding";
840           trace (Printf.sprintf "(%d,%d)" (fst @@ tos_idx ()) (snd @@ tos_idx ()));
841           npush scps ((vm_name uid), curmod.id) (tos_idx ());
842           let _ = curmod.name_id_tick (vm_name uid)
843           in __exec ctxs flags next_ip
844
```

```
845         | IDLE -> trace "idling";
846           __exec ctxs flags next_ip
847
848         | SHOW -> trace "showing";
849           print_string (string_of_value (dspop dss));
850           __exec ctxs flags next_ip
851
852         | PUSH_STACK -> trace "pushing stack";
853           BatDynArray.add dss (dinit ());
854           __exec ctxs flags next_ip
855
856         | SHARE_STACK -> trace "sharing stack";
857           __exec ctxs flags next_ip
858
859         | PUSH_SCOPE -> trace "pushing scope";
860           __exec (inc_scp_count ())
861             {flags with scps = push_scope scps} next_ip
862
863         | POP_SCOPE -> trace "popping scope";
864           (* No GC whatsoever, let OCaml take care of that for me. *)
865           __exec ctxs {flags with scps = ntos scps} next_ip
866
867         | DINT -> trace "setting step debug mode";
868           let rec r () = let input = read_line ()
869             in (match String.sub input 0 2 with
870                   "ss" -> fprintf stderr "scps: %s\n" (sprint_dscope_stack scps); r ()
871                 | "ds" -> fprintf stderr "dss: %s\n" (sprint_dss dss); r ()
872                 | "vi" -> fprintf stderr "val: %s\n" (string_of_value
873                                                         (dval dss
874                                                            ((int_of_string (read_line ())),
875                                                             (int_of_string (read_line ()))))));
876                   r ()
877                 | "ln" -> (try let vidx = resolve_name (int_of_string (read_line ()))
878                                        (int_of_string (read_line ()))
879                                in fprintf stderr "lookup: %s" (string_of_value vidx); r ()
880                            with _ -> fprintf stderr "something wrong happened"; r ())
881                 | "nn" -> __exec ctxs {flags with is_stepping = not flags.is_stepping} next_ip
882                 | _ -> fprintf stderr "something wronger happened"; r ())
883              in r ()
884
885         | EXPORT(ArgLit(VUFixedInt(_n))) -> trace "exporting";
886           let n = vm_name _n
887           in Hashtbl.replace curmod.exs n (dval dss (nlookup scps (n, curmod.id)));
888           __exec ctxs flags next_ip
889
890         | TERMINATE -> trace "terminating";
891           if List.length ctxs <> 1 (* May become a bug? Maybe not? *)
892           then (let tctx = tos ctxs
893                 in let tmod = Hashtbl.find modules tctx.mod_id
894
895                 in trace (sprintf "terminating: %d, ip -> %d"
896                             tctx.mod_id tctx.ret_addr);
897                 dspurge dss;
```

```
                __exec (ntos ctxs)
                  {flags with curmod = tmod;
                              scps = ntos scps}
                  tctx.ret_addr)
          else exit 0

      | _ -> fprintf stderr "Not implemented yet.\n"; exit 0

  in let main = {id = _MAIN_MODULE_ID;
                 insts = insts;
                 imports = Hashtbl.create 64;
                 name_id_tick = name_id_tick_gen ();
                 exs = Hashtbl.create 1}
  in Hashtbl.replace modules _MAIN_MODULE_ID main;
  Hashtbl.replace main.imports _SELF_MODULE_ID _MAIN_MODULE_ID;

  __exec ctxs_ {dss = dinit ();
                is_stepping = false;
                scps = [];
                curmod = Hashtbl.find modules _MAIN_MODULE_ID}
    0;;
```

B.27 vm/t.mli

Code Listing

```
(* vm/t.mli -- Author: Zihang Chen (zc2324) *)
open Stdint;;

type name_t = int;;
type asm_name_t = uint64;;
type asm_name_anno_t = string;;

type module_id_t = int;;
type line_no_t = int;;

type closure_t = (name_t * module_id_t, value_t ref) Hashtbl.t
and type_hint_t = THInt | THAtom | THFixedInt | THUFixedInt | THString
                | THFloat | THList | THPhony
                | THFunction | THNil
                | THTuple | THType
and fun_t = {st: line_no_t;
             mod_id: module_id_t;
             closure: closure_t;
             is_partial: bool}
and value_t = OVInt of Big_int.big_int
            | OVAtom of uint64
            | OVFixedInt of int64
            | OVUFixedInt of uint64
            | OVString of string
            | OVFloat of float
            | OVList of value_t list ref
            | OVPhony
```

```
              | OVFunction of fun_t
              | OVTuple of value_t list ref
              | OVNil
              | OVTypeHint of type_hint_t
```

B.28 vm/tvm.ml

Code Listing

```
(* vm/tvm.ml -- Author: Zihang Chen (zc2324) *)
open Tasm_ast;;
open Batteries;;
open Nvm;;

let src_file_r = ref "";;
let trace_r = ref false;;
let no_warnings_r = ref false;;

let commands = [
  ("-t", Arg.Set(trace_r), "Trace the flow of the instructions");
  ("-nw", Arg.Set(no_warnings_r), "Don't display warnings");
];;

let () = Arg.parse commands (fun fn -> src_file_r := fn)
    "Don't panic!";;

let src_file = !src_file_r;;
let src_inchan = Pervasives.open_in src_file;;
(* let src_content = String.concat "\n" @@ input_list src_inchan;; *)
(* Pervasives.close_in src_inchan;; *)

let () =
  (* let lexbuf = Lexing.from_string src_content *)
  (* in let text = Tasm_parser.asm Tasm_scanner.token lexbuf *)
  (* in exec !trace_r (not !no_warnings_r) *)
  (*   (Array.of_list (match text with Asm(ins) -> ins));; *)
  let insts = Tasm_inv_bytecode.parse_bytecode src_inchan
  in exec !trace_r (not !no_warnings_r) insts;;
```

B.29 vm/vm_t.ml

Code Listing

```
(* vm/vm_t.ml -- Author: Zihang Chen (zc2324) *)
open Tasm_ast;;
open T;;
open Stdint;;
open Dscoping;;
open Nstack;;

type ctx_t = {mod_id: module_id_t; ret_addr: line_no_t; curfun: fun_t;
              scp_count: int;
```

```
               is_tail_recursive_call: bool;
               list_make_stack: value_t list ref list};;
type module_t = {id: module_id_t; insts: line array;
                 exs: (name_t, value_t) Hashtbl.t;
                 imports: (module_id_t, module_id_t) Hashtbl.t;
                 name_id_tick: (name_t -> name_t);
                 (* Map from relative module id to absolute module id. *)}


let print_module m =
  Printf.fprintf stderr "Module {
  id = %d; insts = <insts>; exs = %s;
}" m.id "<exs>";;


(* In tail recursive calls, eval-tail jumps two more lines,
   also fun-arg fetches from current data stack rather than parent
   data stack. *)
type flags_t = {curmod: module_t;
                dss: value_t dstack_t dstack_t;
                scps: scope_t list;
                is_stepping: bool};;

type module_table_t = (module_id_t, module_t) Hashtbl.t;;
```

B.30 vm/wscript

Code Listing

```
# vm/wscript -- Author: Zihang Chen (zc2324)
#! /usr/bin/env python3

def j(l, ext):
    return ' '.join('%s.%s' % (it, ext) for it in l)

mlj = lambda l: j(l, 'ml')
mlij = lambda l: j(l, 'mli')
cmij = lambda l: j(l, 'cmi')
cmoj = lambda l: j(l, 'cmo')
cmxj = lambda l: j(l, 'cmx')

def _c(n, ext):
    return '%s.%s' % (n, ext)

mln = lambda n: _c(n, 'ml')
mlin = lambda n: _c(n, 'mli')
cmin = lambda n: _c(n, 'cmi')
cmon = lambda n: _c(n, 'cmo')
cmxn = lambda n: _c(n, 'cmx')


def build(ctx):
```

```
    mlis = ['t', 'tasm_ast']
    lib_mls = ['exc', 'nstack', 'common', 'config']
    aff_mlis = []
    non_exec_mls = ['dscoping', 'tasm_inv_bytecode']
    some_more_mls0 = ['vm_t']
    some_more_mls1 = ['imp', 'jumps', 'ext', 'arithmetics']
    some_more_mls2 = ['nvm']
    target_mls = ['tvm']

    external_libs = ','.join(ctx.env.TVM_LIBS)
    incl_folder = '%s/src/vm/' % ctx.out_dir
    build_prefix = '%s/' % ctx.out_dir
    # stupid waf!!

    ctx.exec_command([
        'ruby', 'src/tasm/ccg.rb',
        'src/vm', 'src/tasm/scanner.mll.p'
    ])

    ctx.exec_command([
        'ruby', 'src/vm/jmg.rb',
        'src/vm'
    ])

    ctx.exec_command([
        'ruby', 'src/vm/amg.rb',
        'src/vm'
    ])

    obj_ext = cmxn if ctx.options.compile_natively else cmon
    obj_j = cmxj if ctx.options.compile_natively else cmoj

    oc_rule = '${OCAMLFIND} ${OC} ${DEBUG} -package %s'\
              ' -I %s -c -o ${TGT} ${SRC}'\
             % (external_libs, incl_folder,)

    ctx.add_group('vmparser')
    ctx(rule='${MENHIR} --base %s/src/vm/tasm_parser ${SRC}'\
        % ctx.out_dir,
        source='tasm_parser.mly',
        target='tasm_parser.ml tasm_parser.mli')

    ctx(rule='${OCAMLLEX} ${SRC} -o %s${TGT}' % build_prefix,
        source='tasm_scanner.mll',
        target='tasm_scanner.ml')

    ctx(rule=oc_rule,
        source='tasm_ast.mli',
        target=ctx.path.get_bld().find_or_declare('tasm_ast.cmi'))

    ctx(rule=oc_rule,
        source='t.mli',
        target=ctx.path.get_bld().find_or_declare('t.cmi'))
```

```
77
78      def cpl_oc(n, src_ext_f, tgt_ext_f):
79          ctx(rule=oc_rule,
80              source=src_ext_f(n),
81              target=tgt_ext_f(n))
82
83      ctx.add_group('vmlib')
84      for it in lib_mls:
85          cpl_oc(it, mln, obj_ext)
86
87      for it in aff_mlis:
88          cpl_oc(it, mlin, cmin)
89
90      ctx.add_group('vmnon_exec')
91      for it in non_exec_mls:
92          cpl_oc(it, mln, obj_ext)
93
94      ctx.add_group('vmmore_non_exec0')
95      for it in some_more_mls0:
96          cpl_oc(it, mln, obj_ext)
97
98      ctx.add_group('vmmore_non_exec1')
99      for it in some_more_mls1:
100         cpl_oc(it, mln, obj_ext)
101
102     ctx.add_group('vmmore_non_exec2')
103     for it in some_more_mls2:
104         cpl_oc(it, mln, obj_ext)
105
106     def cpl_tgt(n):
107         ctx(rule=oc_rule,
108             source=mln(n),
109             target=obj_ext(n))
110         ctx(rule='${OCAMLFIND} ${OC} ${DEBUG} -package %s -linkpkg'
111                  ' -o %s${TGT} ${SRC}' %\
112                  (external_libs, build_prefix),
113             source=obj_j(lib_mls + non_exec_mls
114                          + some_more_mls0
115                          + some_more_mls1
116                          + some_more_mls2 + [n]),
117             target=n)
118
119     ctx.add_group('vmtgt')
120     for n in target_mls:
121         cpl_tgt(n)
```

B.31 tasm/ccg.rb

Code Listing

```
1  # tasm/ccg.rb -- Author: Zihang Chen (zc2324)
2  =begin
```

```
=begin
This is ccg.py (Towel Assembly Parser generator script).

This script accepts two arguments, the first one for output directory, the
second one for the prototype file for scanner.mll.

This script outputs the following files:
  * tasm_ast.mli
  * tasm_parser.mly
  * tasm_scanner.mll
  * tasm_stringify.ml
  * tasm_bytecode.ml
  * tasm_inv_bytecode.ml
for a minimal TASM parser.
=end

require_relative 'inst'

NUINST = Inst::NUINST
UNINST = Inst::UNINST
BIINST = Inst::BIINST
MAINST = Inst::MAINST

OUTD = ARGV[0]
SCANNER_P = ARGV[1]

def _open(fn, &block)
  open File.join(OUTD, "tasm_#{fn}"), 'w', &block
end

class String
  def in2id
    self.gsub(/-/, '_').upcase
  end
  def id2in
    self.gsub(/_/, '-').downcase
  end
end

def gen_scanner
  _open 'scanner.mll' do |f|
    wl = lambda {|x| f.write x; f.write "\n"}
    content = open(SCANNER_P).read
    wl.call content

    for inst in NUINST + UNINST + BIINST + MAINST
      wl.call "| \"#{inst}\" { #{inst.in2id} }"
    end

    wl.call "| _ as s {
    failwith (Printf.sprintf \"unexpected character `%c'\" s)
}"
  end
end
```

```
56
57  def gen_parser
58    _open 'parser.mly' do |f|
59      wl = lambda {|x| f.write x; f.write "\n"}
60      wl.call "%{
61  open Tasm_ast
62  %}
63
64  %token EOF #{(UNINST + NUINST + BIINST + MAINST).map(&:in2id).join " "}
65
66  %token <Tasm_ast.lit> LITERAL
67  %token <Tasm_ast.label> LABEL
68
69  %start asm
70  %type <Tasm_ast.asm> asm
71
72  %%
73
74  inst:"
75
76      for i in NUINST
77        id = i.in2id
78        wl.call "| #{id} { #{id} }"
79      end
80
81      for i in UNINST
82        id = i.in2id
83
84        if Inst::INST_LABELS.include? i
85          wl.call "| #{id} LABEL { #{id}(ArgLabel($2)) }"
86        end
87
88        wl.call "| #{id} LITERAL { #{id}(ArgLit($2)) }"
89      end
90
91      for i in BIINST
92        id = i.in2id
93        wl.call "| #{id} LITERAL LITERAL { #{id}(ArgLit($2), ArgLit($3)) }"
94      end
95
96      for i in MAINST
97        id = i.in2id
98        wl.call "| #{id} nonempty_list(LITERAL)
99      { #{id}(List.map (fun x -> ArgLit(x)) $2) }
100 "
101     end
102
103     wl.call "line: list(LABEL) inst { Line($1, $2) }
104
105 asm: list(line) EOF { Asm($1) }"
106   end
107 end
108
```

```
def gen_ast
  _open 'ast.mli' do |f|
    wl = lambda {|x| f.write x; f.write "\n"}

    gen_defs = lambda {|fmt, defs|
      if defs.empty?
        ''
      else
        defs.map {|x| fmt % x.in2id}.join " | "
      end
    }

    wl.call "open Stdint;;

type lit = VString of string
         | VAtom of uint64
         | VInt of Big_int.big_int
         | VFixedInt of int64
         | VUFixedInt of uint64
         | VFloat of float;;

type label = Label of string;;

type arg = ArgLit of lit
         | ArgLabel of label;;

type inst =
  #{NUINST.map(&:in2id).join " | "}
#{unless UNINST.empty? then " | " else "" end}#{gen_defs.call "%s of arg", UNINST}
#{unless BIINST.empty? then " | " else "" end}#{gen_defs.call "%s of arg * arg", BIINST}
#{unless MAINST.empty? then " | " else "" end}#{gen_defs.call "%s of arg list", MAINST};;

type line = Line of label list * inst
          | CLine of label list * inst option;;

type asm = Asm of line list;;
"
  end
end

def gen_stringify
  _open 'stringify.ml' do |f|
    wl = lambda {|x| f.write x; f.write "\n"}

    wl.call <<LINEEND
open Stdint
open Tasm_ast

let p_label = function Label(l) -> l;;

let p_labels ls = String.concat " " (List.map p_label ls);;

let p_arg = function
```

```
    ArgLabel(l) -> p_label l
  | ArgLit(v) -> (match v with
      VString(s) -> Printf.sprintf "'%s'" s
    | VAtom(a) -> Printf.sprintf "%sa" @@ Uint64.to_string a
    | VInt(i) -> Printf.sprintf "%sl" @@ Big_int.string_of_big_int i
    | VFixedInt(i) -> Int64.to_string i
    | VUFixedInt(u) -> Printf.sprintf "%su" @@ Uint64.to_string u
    | VFloat(f) -> string_of_float f);;

let _p_inst inst arg = inst ^ " " ^ (p_arg arg);;
let _p_inst_ba inst arg1 arg2 = inst ^ " " ^ (p_arg arg1) ^ " " ^ (p_arg arg2);;
let _p_inst_ma inst args =
  inst ^ " " ^ (String.concat " " (List.map p_arg args));;
let _p_inst_na inst = inst;;

let p_inst =
  function
  #{NUINST.map {|x| "#{x.in2id} -> _p_inst_na \"#{x}\""}.join " | "}
  #{unless UNINST.empty? then " | " else "" end}#{UNINST.map {|x|
 "#{x.in2id}(arg) -> _p_inst \"#{x}\" arg"}.join " | "}
  #{unless BIINST.empty? then " | " else "" end}#{BIINST.map {|x|
 "#{x.in2id}(arg1, arg2) -> _p_inst_ba \"#{x}\" arg1 arg2"}.join " | "}
  #{unless MAINST.empty? then " | " else "" end}#{MAINST.map {|x|
 "#{x.in2id}(args) -> _p_inst_ma \"#{x}\" args"}.join " | "}

let p_line = function
  Line(ls, inst) -> (p_labels ls) ^ " " ^ (p_inst inst)
| CLine(ls, maybe_inst) -> (p_labels ls) ^
    (match maybe_inst with
       Some(inst) -> " " ^ (p_inst inst)
     | None -> "");;

let p_asm = function
  Asm(lines) -> String.concat "\n" (List.map p_line lines);;
LINEEND
  end
end

NUBIN = Hash[Inst::NUALL]
UNBIN = Hash[Inst::UNALL]
BIBIN = Hash[Inst::BIALL]
MABIN = Hash[Inst::MAALL]

MAGIC_4242 = "BB"
MAGIC_DATA_END = '\2550\128\255'

def gen_bytecode_compiler
  _open 'bytecode.ml' do |f|
    wl = lambda {|x| f.write x; f.write "\n"}

    wl.call <<LINEEND
open Stdint;;
open Tasm_ast;;
```

```
215
216 let _MAGIC_4242 = Bytes.of_string "#{MAGIC_4242}";;
217 let _MAGIC_DATA_END = Bytes.of_string "#{MAGIC_DATA_END}";;
218
219 let nil = Char.chr 0;;
220
221 let data = Hashtbl.create 512;;
222 let __x () =
223   let table = Hashtbl.create 512
224   in let _a = [|0|]
225   in let tick () = _a.(0) <- _a.(0) + 1; _a.(0)
226   in (fun v -> try
227          Hashtbl.find table v
228        with Not_found ->
229          let x = tick ()
230          in Hashtbl.replace table v x;
231          x),
232      (fun v -> Hashtbl.find table v);;
233
234 let vid, _ = __x ();;
235
236 let mark x = Bytes.cat (Bytes.of_string x);;
237
238 let _8B () = Bytes.init 8 (fun _ -> nil);;
239 let _4B () = Bytes.init 4 (fun _ -> nil);;
240
241 let uint64_to_bytes x = let _content = _8B ()
242   in Uint64.to_bytes_little_endian x _content 0;
243   _content;;
244
245 let int64_to_bytes x = let _content = _8B ()
246   in Int64.to_bytes_little_endian x _content 0;
247   _content;;
248
249 let string_to_bytes x = let _len = _4B ()
250   in Uint32.to_bytes_little_endian (Uint32.of_int (String.length x))
251     _len 0;
252   Bytes.cat _len (Bytes.of_string x);;
253
254 let store =
255   let save bytes =
256     let id_ = vid bytes
257     in (if Hashtbl.mem data id_
258         then () else Hashtbl.replace data id_ bytes);
259     id_
260   in function
261     ArgLabel(l) -> failwith "Please assemble the IR first."
262   | ArgLit(v) -> (match v with
263         VString(s)
264         -> let str_b = mark "s" (string_to_bytes s)
265         in save str_b
266       | VAtom(a)
267         -> let atom_b = mark "a" (uint64_to_bytes a)
```

```
          in save atom_b
        | VInt(i)
          -> let int_b = mark "i" (i
                                   |> Big_int.string_of_big_int
                                   |> string_to_bytes)
          in save int_b
        | VFixedInt(i)
          -> let fint_b = mark "f" (int64_to_bytes i)
          in save fint_b
        | VUFixedInt(u)
          -> let ufint_b = mark "u" (uint64_to_bytes u)
          in save ufint_b
        | VFloat(f)
          -> let float_b = mark "n" (* stands for NaN *)
                 (f |> string_of_float |> string_to_bytes)
          in save float_b);;

let _b_inst_na x = uint64_to_bytes (Uint64.of_int x);;

let _b_inst x arg_ =
  let _inst_bin = _b_inst_na x
  in let arg_id = store arg_
  in let _content = _8B ()
  in let _t = uint64_to_bytes (Uint64.of_int arg_id)
  in Bytes.blit _inst_bin 0 _content 0 1;
  Bytes.blit _t 0 _content 1 7;
  _content;;

let _b_inst_ba x arg1 arg2 =
  let _inst_bin = _b_inst_na x
  in let arg1_id = store arg1
  in let arg2_id = store arg2
  in let _content = _8B ()
  in let _1 = uint64_to_bytes (Uint64.of_int arg1_id)
  in let _2 = uint64_to_bytes (Uint64.of_int arg2_id)
  in Bytes.blit _inst_bin 0 _content 0 1;
  Bytes.blit _1 0 _content 1 4;
  Bytes.blit _2 0 _content 5 3;
  _content;;

let b_inst =
  function
  #{NUINST.map {|x| "#{x.in2id} -> _b_inst_na #{NUBIN[x]}"}.join " | "}
  #{unless UNINST.empty? then " | " else "" end}#{UNINST.map {|x|
 "#{x.in2id}(arg) -> _b_inst #{UNBIN[x]} arg"}.join " | "}
  #{unless BIINST.empty? then " | " else "" end}#{BIINST.map {|x|
 "#{x.in2id}(arg1, arg2) -> _b_inst_ba #{BIBIN[x]} arg1 arg2"}.join " | "}
  #{unless MAINST.empty? then " | " else "" end}#{MAINST.map {|x|
 "#{x.in2id}(args) -> _b_inst_ma #{MABIN[x]} args"}.join " | "}

let b_line = function
  Line(_, inst) -> b_inst inst
| CLine(ls, maybe_inst) ->
```

```
    (match maybe_inst with
       Some(inst) -> b_inst inst
     | None -> Bytes.of_string "");;

let b_asm = function
    Asm(lines) ->
    let lines_b = (List.fold_left (fun acc x -> Bytes.cat acc x)
                     (Bytes.of_string "")
                     (List.map b_line lines))
    in let data_seg =
      let sorted_data =
        List.sort (fun x y -> Pervasives.compare (fst x) (fst y))
        @@ Hashtbl.fold (fun k v acc -> (k, v)::acc) data []
      in List.fold_left (fun acc x -> Bytes.cat acc (snd x))
        _MAGIC_4242
        sorted_data
    in Bytes.cat
      (Bytes.cat data_seg _MAGIC_DATA_END)
      lines_b;;
LINEEND
  end
end

def gen_inv_bytecode
  _open 'inv_bytecode.ml' do |f|
    wl = lambda {|x| f.write x; f.write "\n"}

    wl.call <<LINEEND
open Stdint;;
open Tasm_ast;;
open Batteries;;
open BatIO;;
open BatEnum;;


let next_byte_str input = input |> read_byte |> char_of_int |> String.make 1;;

let next_string_raw input n =
  fold (fun acc _ -> acc ^ (next_byte_str input)) "" (1 -- n);;

let next_int input =
  let xs = next_string_raw input 4 |> Bytes.of_string
  in Uint32.of_bytes_little_endian xs 0 |> Uint32.to_int;;

let next_u64 input =
  let xs = next_string_raw input 8 |> Bytes.of_string
  in Uint64.of_bytes_little_endian xs 0;;

let next_string input =
  let n = next_int input
  in next_string_raw input n;;

let recover_data input =
```

```
  let __tick = [|0|]
  in let tick () = __tick.(0) <- __tick.(0) + 1; __tick.(0)

  in let rec next_data acc =
    let indicator = next_byte_str input
    in if indicator = "\\255"
    then if next_string_raw input 3 = "0\\128\\255"
      then acc
      else failwith "Unknown data end magic."
    else let arg = match indicator with
          "s" -> ArgLit(VString(next_string input))
        | "a" -> ArgLit(VAtom(next_u64 input))
        | "i" -> ArgLit(VInt(input |> next_string |> Big_int.of_string))
        | "f" -> ArgLit(VFixedInt(next_u64 input |> Uint64.to_int64))
        | "u" -> ArgLit(VUFixedInt(next_u64 input))
        | "n" -> ArgLit(VFloat(next_string input |> float_of_string))
        | _ -> failwith "Corrupted bytecode."
      in Hashtbl.replace acc (tick ()) arg;
      next_data acc

  in next_data (Hashtbl.create 512);;

let _OPCODE_MASK = Uint64.of_int 0xff;;
let _ARG_SHIFT = 8;;
let _ARG1_SHIFT = 8;;
let _ARG2_SHIFT = 40;;
(*
   0 -- 7|8 -- 15|16 -- 23|24 -- 31|32 -- 39|40 -- 47|48 -- 55|56 -- 63
   opcode unused
   opcode arg
   opcode arg1                                arg2
 *)

let inv_b_inst data i =
  let opcode = Uint64.to_int (Uint64.logand i _OPCODE_MASK)
  in let iarg = Uint64.shift_right_logical i _ARG_SHIFT |> Uint64.to_int
  in let iarg1 = Uint64.shift_right_logical i _ARG1_SHIFT |> Uint64.to_int
  in let iarg2 = Uint64.shift_right_logical i _ARG2_SHIFT |> Uint64.to_int
  in let retrieve i = Hashtbl.find data i

  in match opcode with
    #{NUINST.map {|x| "#{NUBIN[x]} -> #{x.in2id}"}.join " | "}
 #{unless UNINST.empty? then " | " else "" end}#{UNINST.map {|x|
"#{UNBIN[x]} -> #{x.in2id}(retrieve iarg)"}.join " | "}
 #{unless BIINST.empty? then " | " else "" end}#{BIINST.map {|x|
"#{BIBIN[x]} -> #{x.in2id}(retrieve iarg1, retrieve iarg2)"}.join " | "}
  | _ -> failwith "Unrecognized opcode.";;

let parse_bytecode inchan =
  let input = input_channel inchan
  in let insts = ref []
  in if next_string_raw input 2 = "BB"
  then let data = recover_data input
```

```
    in let rec __x () =
         try let b_inst = next_u64 input
           in insts := (Line([], inv_b_inst data b_inst))::(!insts);
           __x ()
         with BatIO.No_more_input ->
           insts := List.rev (!insts);
    in __x (); Array.of_list (!insts)
  else failwith "Invalid Towel bytecode file.";;
LINEEND
  end
end

if __FILE__ == $0
  gen_scanner
  gen_parser
  gen_ast
  gen_stringify
  gen_bytecode_compiler
  gen_inv_bytecode
end
```

B.32 tasm/inst.rb

Code Listing

```
# tasm/inst.rb -- Author: Zihang Chen (zc2324)
=begin
Definition for the instructions of Towel Assembly Language.

See also the Towel Assembly Language Manual.
=end

require 'set'

module Inst
  nullary_instructions_all = [
    ['push-scope', 0x01],
    ['pop-scope', 0x02],
    ['share-scope', 0x03],

    ['end-list', 0x13], ['end-tuple', 0x14],
    ['pop', 0x18], ['push-phony', 0x1d],
    ['car', 0x5e], ['cdr', 0x5f], ['list-empty', 0x61],
    ['cons', 0x60],
    ['unpack', 0x1b],
    ['push-stack', 0x20], ['share-stack', 0x21], ['pop-stack', 0x22],
    ['ret', 0x24], ['shared-ret', 0x25],
    ['show', 0x67], ['read', 0x68],
    ['dint', 0x71],
    ['type', 0x72],
    ['not-implemented', 0xff],
    ['idle', 0x0],
    ['terminate', 0xfe],
```

```
      ['push-lnil', 0x11],
      ['push-tnil', 0x12],
      ['reverse', 0x1a],
      ['pack', 0x1c],
      ['invoke', 0x27],
      ['add', 0x50], ['sub', 0x51], ['mul', 0x52],
      ['div', 0x53], ['mod', 0x55], ['equ', 0x56],
      ['and', 0x57], ['or', 0x58], ['xor', 0x59], ['not', 0x5a],
      ['shl', 0x5b], ['shr', 0x5c], ['lshr', 0x5d], ['dup', 0x19],
      ['load-ext', 0xf1], ['extcall', 0xf0],
      ['install', 0x29], ['tuple-at', 0xf3], ['sweep', 0x2a],
    ]

    unary_instructions_all = [
      ['bind', 0x04], ['fun-arg', 0x05],
      ['jump', 0x30],
      ['push-lit', 0x15],
      ['call', 0x28],
      ['push-fun', 0x10],
      ['closure', 0x26],
      ['import', 0x70],
      ['export', 0x7f],
    ]

    binary_instructions_all = [
      ['eval-and-push', 0x17],
      ['push-name', 0x16],
      ['eval-tail', 0x23],
    ]

    multiarity_instructions_all = []

    multiarity_instructions = [
    ]

    inst_that_supports_label_as_arguments = Set.new [
      'jump', 'match', 'hmatch'
    ]

    for i in ['push']
      inst_that_supports_label_as_arguments.add "#{i}-fun"
    end
    inst_that_supports_label_as_arguments.add "call"

    for s in [['gez', 1],
              ['gz', 2],
              ['lez', 3],
              ['lz', 4],
              ['ez', 5],
              ['nez', 6],
              ['t', 7],
              ['f', 8],
              ['e', 9],
```

```
              ['ne', 10]]
    i, j = s
    unary_instructions_all.push ["j#{i}", 0x30 + j]
    inst_that_supports_label_as_arguments.add "j#{i}"

    unary_instructions_all.push ["hj#{i}", 0x40 + j]
    inst_that_supports_label_as_arguments.add "hj#{i}"
  end

  for s in [['fint', 2], ['ufint', 3], ['int', 4], ['float', 5], ['str', 6]]
    i, j = s
    nullary_instructions_all.push ["to-#{i}", 0x60 + j]
  end

  nullary_instructions = nullary_instructions_all.map {|x| x[0]}
  nullary_instructions_bin = nullary_instructions_all.map {|x| x[1]}

  unary_instructions = unary_instructions_all.map {|x| x[0]}
  unary_instructions_bin = unary_instructions_all.map {|x| x[1]}

  binary_instructions = binary_instructions_all.map {|x| x[0]}
  binary_instructions_bin = binary_instructions_all.map {|x| x[1]}

  NUINST = nullary_instructions
  UNINST = unary_instructions
  BIINST = binary_instructions
  MAINST = multiarity_instructions
  NUALL = nullary_instructions_all
  UNALL = unary_instructions_all
  BIALL = binary_instructions_all
  MAALL = multiarity_instructions_all
  INST_LABELS = inst_that_supports_label_as_arguments
end
```

B.33 tasm/scanner.mll.p

Code Listing

```
(* tasm/scanner.mll.p -- Author: Zihang Chen (zc2324) *)
{
open Tasm_ast
open Tasm_parser
open Stdint

let strip_mod s =
  let len = String.length s
  in String.sub s 0 (len - 1)

let strip_sign s =
  let len = String.length s
  in let first = String.sub s 0 1
  in if (first = "+") || (first = "-")
  then String.sub s 1 @@ len - 1
```

```
    else s
}

let _WHITESPACE = [' ' '\t']
let _NEWLINE = '\n' | '\r' | "\r\n"
let _SQUOTE = '\''
let _DQUOTE = '"'

let string_char = [^ '\\' '\'']
let string_esc_charseq = '\\' ['\'' '\\' 'n' 'r' 'b' ' ' 't']
let string_item = string_char | string_esc_charseq
let string_lit = _SQUOTE string_item* _SQUOTE
(* From python lexical analysis
   https://docs.python.org/3/reference/lexical_analysis.html#string-and-bytes-literals*)

let digit = ['0'-'9']
let hexdigit = ['0'-'9' 'a'-'f' 'A'-'F']
let bindigit = ['0' '1']
let signed = ['+' '-']
let _int_lit = (("0d"? digit+)
              | ("0x" hexdigit+)
              | ("0b" bindigit+))
let fint_lit = signed? _int_lit
let int_lit = signed? digit+ ['L' 'l']
let ufint_lit = '+'? _int_lit ['U' 'u']
let atom_lit = _int_lit ['A' 'a']

let dot = '.'
let int = digit+
let frac = digit+
let exp = 'e' signed? int
let dot_float = ((dot frac) | (int dot frac)) exp?
let exp_float = int (dot frac)? exp
let float_lit = signed? (dot_float | exp_float)

let label_lit = ":" ['0'-'9' 'a'-'z' 'A'-'Z' '-']+ "!"?

rule token = parse
| _WHITESPACE+ { token lexbuf }
| _NEWLINE { Lexing.new_line lexbuf; token lexbuf }

| eof { EOF }

| _DQUOTE [^ '"' '\n' '\r']* _DQUOTE { token lexbuf } (* comments *)

| label_lit as a { LABEL(Tasm_ast.Label(a)) }
| string_lit as str {
    LITERAL(VString(String.sub str 1 (String.length str - 2)))
  }
| fint_lit as i {
    LITERAL(VFixedInt(Int64.of_string i))
  }
| int_lit as i {
```

```
    LITERAL(VInt(Big_int.big_int_of_string @@ strip_mod i))
  }
| ufint_lit as i {
    LITERAL(VUFixedInt(Uint64.of_string
                         (i |> strip_sign |> strip_mod)))
  }
| atom_lit as i {
    LITERAL(VAtom(Uint64.of_string (i |> strip_mod)))
  }
| float_lit as f {
    LITERAL(VFloat(float_of_string f))
  }
```

B.34 towelibs/ext_random.ml

Code Listing

```
(* towelibs/ext_random.ml -- Author: Zihang Chen (zc2324) *)
open T;;
open Nstack;;
open Ext;;
open Stdint;;

let rndfail msg = failwith (Printf.sprintf "ExtRandom> %s" msg);;

module ExtRandom:TowelExtTemplate =
struct
  let extcall cn dss = match cn with
      1 -> Random.self_init ()
    | 2 ->
      let seed = match dspop dss with
          OVFixedInt(i) -> Int64.to_int i
        | OVUFixedInt(i) -> Uint64.to_int i
        | OVInt(i) -> Big_int.int_of_big_int i
        | OVFloat(f) -> int_of_float f
        | _ -> rndfail "invalid random seed"
      in Random.init seed
    | 3 -> dspush dss (OVFloat(Random.float 1.0))
    | _ -> rndfail "unrecognized ExtRandom cnumber"
  end

let () = __ext__ := Some(module ExtRandom:TowelExtTemplate);;
```

B.35 towelibs/ext_tk.ml

Code Listing

```
(* towelibs/ext_tk.ml -- Author: Zihang Chen (zc2324) *)
open Tk;;
open T;;
open Ext;;
open Nstack;;
```

```
6
7  let top = ref None;;
8
9  let widgets:(int, Widget.toplevel Widget.widget) Hashtbl.t = Hashtbl.create 512;;
10
11 let tkfail msg = failwith (Printf.sprintf "TK failure: %s.\n" msg);;
12
13 module SimpleTk : TowelExtTemplate =
14 struct
15   let extcall cn dss = match cn with
16       1 -> top := Some(openTk ())
17     | 2 -> mainLoop ()
18     | 3 -> closeTk ()
19     | 4 -> update ()
20     | 5 -> let s = appname_get ()
21       in dspush dss (OVString(s))
22     | 6 -> let s = match (dspop dss) with
23           OVString(x) -> x
24         | _ -> tkfail "unsupported data type for appname_set"
25       in appname_set s
26     | _ -> tkfail "unimplemented call number"
27 end
28
29 let () = __ext__ := Some(module SimpleTk : TowelExtTemplate);;
```

B.36 towelibs/random.t

Code Listing

```
"towelibs/random.t -- Author: Zihang Chen (zc2324)"

import '.w' \

bind !>> !>>\.w`

also ^~ ('ext_random.cmo' !>ext\.w)
"The ext module handle."

also ~seed fun` ~s, (~s 2u ^~ !>>)
also ~useed fun`, (1u ^~ !>>)
also ~~ fun`, (3u ^~ !>>)
then export ~seed ~useed ~~ @
```

B.37 towelibs/simple-tk.t

Code Listing

```
"towelibs/simple-tk.t -- Author: Zihang Chen (zc2324)"

import '.w' \

bind ^~ ('ext_tk.cmo' !>ext\.w)
```

```
also !>> !>>\.w`

also >>tk fun`, (1u ^~ !>>)
also ~~~ fun`, (2u ^~ !>>)
also <<tk fun`, (3u ^~ !>>)

also !set-tk-appname fun` ~s, (~s 6u ^~ !>>)

then export >>tk ~~~ <<tk !set-tk-appname @
```

B.38 towelibs/std.t

Code Listing

```
"towelibs/std.t -- Author: Zihang Chen (zc2324)"

import '.w' \

bind + fun` ~1 ~2, (~1 ~2 ..+\.w)
also - fun` ~1 ~2, (~1 ~2 ..-\.w)
also * fun` ~1 ~2, (~1 ~2 ..*\.w)
also / fun` ~1 ~2, (~1 ~2 ../\.w)
also % fun` ~1 ~2, (~1 ~2 ..%\.w)
also = fun` ~1 ~2, (~1 ~2 ..=\.w)

also > fun` ~1 ~2, (~1 ~2 - if>0 true, false)
also >= fun` ~1 ~2, (~1 ~2 - if>=0 true, false)
also < fun` ~1 ~2, (~1 ~2 - if<0 true, false)
also <= fun` ~1 ~2, (~1 ~2 - if<=0 true, false)
also <> fun` ~1 ~2, (~1 ~2 = ift false, true)

also :and fun` ~1 ~2, (~1 ~2 ..and\.w)
also :or fun` ~1 ~2, (~1 ~2 ..or\.w)
also :not fun` ~, (~ ..not\.w)
also :xor fun` ~1 ~2, (~1 ~2 ..xor\.w)
also :shl fun` ~x ~n, (~x ~n ..shl\.w)
also :shr fun` ~x ~n, (~x ~n ..shr\.w)
also :lshr fun` ~x ~n, (~x ~n ..lshr\.w)

also ~fint fun` ~1, (~1 ..2fint\.w)
"Use tilde as prefix for all the conversion functions."
also ~ufint fun` ~1, (~1 ..2ufint\.w)
also ~int fun` ~1, (~1 ..2int\.w)
also ~float fun` ~1, (~1 ..2float\.w)
also ~str fun` ~1, (~1 ..2str\.w)

also ^? .^?\.w`

also !print .!print\.w`
"Use exclamation mark as prefix for functions with side-effects."
"Use .! as prefix for functions with serious stack-effects."
also !println .!println\.w`
also !read .!read\.w`
```

```
also !!pop .!pop\.w`
also !!dup .!dup\.w`

also $$ $$\.w`

also #hd fun` ~1, (~1 ..hd\.w)
also <# #hd`
"Use # as prefix for list/tuple functions."
also #tl fun` ~1, (~1 ..tl\.w)
also #> #tl`
also #cons fun` ~1 ~2, (~1 ~2 ..cons\.w)
also <#> #cons`
also ?#empty fun` ~1, (~1 .?empty\.w)
also ?# ?#empty`
also #n fun` ~n ~t, (~t ~n ..tuple-at\.w)
also #1 fun` ~1, (0 ~1 #n)
also #2 fun` ~1, (1 ~1 #n)
also #3 fun` ~1, (2 ~1 #n)

also !!pack .!pack\.w`
also !!unpack .!unpack\.w`

also /id fun` ~1, ~1
"/\ -> lambda, since \ is reserved, we use /"
also /foldl fun` ~p ~l ~f, (
  ~l .?empty\.w ift ~p,
  (~p ~l ..hd\.w ~f "accumulated value"
   ~l ..tl\.w "rest of the list"
   ~f`
   /foldl@))
also /foldr fun` ~p ~l ~f, (
  ~l .?empty\.w ift ~p,
  (~p ~l ..tl\.w ~f` /foldr
   ~l ..hd\.w
   ~f))
also #rev fun` ~1, ([] ~1 fun` ~acc ~x, (~x ~acc ..cons\.w) /foldl@)
also <<# #rev`
also |#| fun` ~1 ~2, (~2 ~1 #rev fun` ~acc ~x, (~x ~acc ..cons\.w) /foldl@)
also #concat |#|`
also /map fun` ~l ~f, (
  [] ~l fun` ~acc ~x, (~x ~f ~acc ..cons\.w) /foldl #rev@)
also /filter fun` ~l ?pred, (
  [] ~l
  fun` ~acc ~x, (~x ?pred ift ~acc,
                 (~x ~acc ..cons\.w))
  /foldl #rev@)

also /arg-rot-n fun` ~f ~n, (
  {~n .!pack\.w fun ~args, (~args ..tl\.w [(~args ..hd\.w)] #concat)
   .!unpack\.w ~f})
also /arg-rot3 (3 /arg-rot-n)
also /flip (2 /arg-rot-n)
```

```
also #len fun` ~1, (0u ~1 fun` ~acc _, (~acc 1u ..+\.w) /foldl@)
also !# #len`

also /apply fun` ~args ~f, (~args .!unpack\.w ~f)

also .//foldl (/foldl` /arg-rot3)
also .///foldl (/foldl` /arg-rot3 /arg-rot3)

also #!vargs {-1 .!pack\.w}
also /!vfoldl {#!vargs .//foldl}
also #!vconcat {#!vargs #concat` [] .///foldl}

also !invoke .!invoke\.w`

also ~idle ..idle\.w`

then export + - * / % =
:and :or :xor :shl :shr :lshr
~fint ~ufint ~int ~float ~str !print
!println #hd #tl #cons ?#empty #1 #2 #3 /id /foldl #rev /map /filter
/flip /foldr |#| #concat <# #> <#> ?# <<# !# #len /apply !read > >= < <= <>
:not !!pop !!dup !!pack !!unpack $$ ^? #n ~idle !invoke /!vfoldl #!vconcat
#!vargs /arg-rot-n /arg-rot3 @
```

B.39 towelibs/std_gen.rb

Code Listing

```ruby
# towelibs/std_gen.rb -- Author: Zihang Chen (zc2324)
# This file generates raw wrapper functions for the instructions.
# Most of the raw wrappers have their corresponding partial
# applicable high-level wrappers in module Std.
# However, wrappers for some of the stack instructions are not
# high-level available, because they are designed to work on
# primitive stacks (`pop', for example).

require 'stringio'

def gen_make_inline_fun(w, e, st_line_=2, nid_=1)
  st_line = st_line_
  nid = nid_

  lambda do |name, body|
    end_line = body.size + st_line + 5
    w.puts "push-fun #{(st_line + 4).to_s}u"
    w.puts "bind #{nid}u \"#{name}\""
    w.puts "export #{nid}u"
    w.puts "jump #{end_line}u"
    body.each {|inst| w.puts inst}
    w.puts "shared-ret"

```

```
24      e.puts "(#{name} #{nid}u)"
25
26      st_line = end_line
27      nid += 1
28    end
29  end
30
31  if __FILE__ == $0
32    w = StringIO.new
33    e = StringIO.new
34
35    make_inline_fun = gen_make_inline_fun w, e
36
37    w.puts 'push-scope'
38    w.puts 'push-stack'
39
40    for i in [['add', '..+'],
41              ['sub', '..-'],
42              ['mul', '..*'],
43              ['div', '../'],
44              ['mod', '..%'],
45              ['equ', '..='],
46              ['and', '..and'],
47              ['or', '..or'],
48              ['xor', '..xor'],
49              ['shl', '..shl'],
50              ['shr', '..shr'],
51              ['lshr', '..lshr'],
52              ['not', '..not']]
53      instn, funn = i
54
55      make_inline_fun.call "#{funn}", ["#{instn}"]
56    end
57
58    types =[['fint', 'FInt'], ['ufint', 'UFInt'],
59            ['int', 'Int'], ['float', 'Float'], ['str', 'Str']]
60    for i in types
61      make_inline_fun.call"..2#{i[0]}", ["to-#{i[0]}"]
62    end
63
64    make_inline_fun.call '.!print', ['show']
65    make_inline_fun.call '.!println', ['show', "push-lit '\n'", 'show']
66    make_inline_fun.call '.!read', ['read']
67    make_inline_fun.call '.!dup', ['dup']
68    make_inline_fun.call '.!pop', ['pop']
69    make_inline_fun.call '.!rev', ['reverse']
70    make_inline_fun.call '.!invoke', ['invoke']
71    make_inline_fun.call '.!unpack', ['unpack']
72    make_inline_fun.call '.!pack', ['pack']
73    make_inline_fun.call '..hd', ['car']
74    make_inline_fun.call '..tl', ['cdr']
75    make_inline_fun.call '..cons', ['cons']
76    make_inline_fun.call '.?empty', ['list-empty']
```

```
  make_inline_fun.call '..tuple-at', ['tuple-at']
  make_inline_fun.call '.^?', ['type']

  make_inline_fun.call '.#lbegin', ['push-lnil']
  make_inline_fun.call '.#lend', ['end-list']
  make_inline_fun.call '.#tbegin', ['push-tnil']
  make_inline_fun.call '.#tend', ['end-list']

  make_inline_fun.call '!>ext', ['load-ext']
  make_inline_fun.call '!>>', ['extcall']

  make_inline_fun.call '..idle', ['idle']
  make_inline_fun.call '$$', ['push-phony']
  make_inline_fun.call '***', ['dint']

  w.puts 'terminate'

  File.open '.w.l', 'w' do |f| f.puts w.string end
  File.open '.w.e', 'w' do |f| f.puts e.string end
end
```

B.40 towelibs/wscript

Code Listing

```
# towelibs/wscript -- Author: Zihang Chen (zc2324)

def build(ctx):
    ctx.exec_command(['ruby', 'std_gen.rb'])
```

B.41 tests/__foldr.t

Code Listing

```
"tests/__foldr.t -- Author: Zihang Chen (zc2324)"

import 'std' @

([] [1 2 3 4 5 6 7 8] fun` ~acc ~x, (0 ~x - ~acc #cons) /foldr !println)
```

B.42 tests/__read.t

Code Listing

```
"tests/__read.t -- Author: Zihang Chen (zc2324)"

import 'std' @

(!read !print ', hello world!' !println)
```

B.43 tests/__tk.t

Code Listing

```
"tests/__tk.t -- Author: Zihang Chen (zc2324)"

import 'simple-tk' @

(>>tk
 'Hello, world by the Towel programming language!' !set-tk-appname ~~~
 <<tk)
```

B.44 tests/adv_obj.t

Code Listing

```
"tests/adv_obj.t -- Author: Zihang Chen (zc2324)"
"---
Some object-oriented programming sh**.
!>>
!<< Circle
Rectangle
5541.76476
2
"

import 'std' @

bind >>send !invoke`

also Shape ,\ ~type,
  bind __type ~type
  also Type fun`, #1
  also Area #2`
  then [\__type [\Type` Area`]]

then bind __meta-Shape ('' Shape)
     also :type (__meta-Shape #2 #1)
     also :area (__meta-Shape #2 #2)

     also Circle ,\ Radius,
       bind __radius Radius
       also ~super ('Circle' Shape)

       then bind Type (~super #1)
            also Area ,\, (__radius __radius 3.14159 * *)

            then [\Type` Area`]

      also Rectangle ,\ Width Height,
        bind __width Width
        also __height Height
```

```
        also ~super ('Rectangle' Shape)

        then bind Type (~super #1)
             also Area ,\, (__width __height *)

             then [\Type` Area`]

      then bind ~my-circle (42. Circle)
           also ~my-rectangle (1 2 Rectangle)
   then (~my-circle :type >>send !println
         ~my-rectangle :type >>send !println
 ~my-circle :area >>send !println
 ~my-rectangle :area >>send !println)
```

B.45 tests/bind.t

Code Listing

```
"tests/bind.t -- Author: Zihang Chen (zc2324)"
"---
Tests if nested bind and also works.
!>>
!<< 3
-1
5
3
10
-2
9
3
12
11
"

import 'std' \

bind A fun` B,
  bind C fun` D,
    bind E (1 2 -\Std)
    then (((E !println\Std D !println\Std B !println\Std 10)))
  also F fun` D,
    bind E (5 7 -\Std)
    then (E !println\Std D !println\Std B !println\Std 12)
  then (B !println\Std 5 C !println\Std 9 F !println\Std 11)
then (3 A !println\Std)
```

B.46 tests/functional.t

Code Listing

```
"tests/functional.t -- Author: Zihang Chen (zc2324)"
```

```
"---
Tests some of the functional functions.
!>>
!<< 2
[41 40]
-123
[1 3 false]
-1
"
import 'std' @

bind :neg- (-` /flip)
then (1 3 :neg- !println)

bind :dec-1 (1 -)
then ([42 41] :dec-1` /map !println
      0 [42 41 40] -` /foldr !println)

bind ?not-true ,\ ~x, (~x ift true, false)
then ([1 3 true false true] ?not-true` /filter !println)

([1 2] -` /apply !println)
```

B.47 tests/gcd.t

Code Listing

```
"tests/gcd.t -- Author: Zihang Chen (zc2324)"
"Greatest common divisor test case.
---
Computes the greatest common divisor of 42 and 24.
Mainly tests tail recursive calls.
!>>
!<< 6
"

import 'std' @

bind GCD fun` A B,
  (A B - if=0 A,
     if>0 (A B - B GCD@),
     if<0 (A B A - GCD@),
       ~idle)
then (42 24 GCD !println)
```

B.48 tests/idle.t

Code Listing

```
"tests/idle.t -- Author: Zihang Chen (zc2324)"
"---
Tests the idle instruction.
```

```
!>>
!<< "

import '.w' @

..idle
```

B.49 tests/ift-iff.t

Code Listing

```
"tests/ift-iff.t -- Author: Zihang Chen (zc2324)"
"---
Tests ift and iff instruction. And other ifs.
!>>
!<< 12211221"

import 'std' \
import '.w' @

($$ ife (1 !print\Std), (2 !print\Std))
($$ ifne (1 !print\Std), (2 !print\Std))
(1 2 3 ife (1 !print\Std), (2 !print\Std))
(1 2 3 ifne (1 !print\Std), (2 !print\Std))

(true ift (1 !print\Std), (2 !print\Std)) .!pop\.w
(false ift (1 !print\Std), (2 !print\Std)) .!pop\.w
(true iff (1 !print\Std), (2 !print\Std)) .!pop\.w
(false iff (1 !print\Std), (2 !print\Std)) .!pop\.w
```

B.50 tests/import.t

Code Listing

```
"tests/import.t -- Author: Zihang Chen (zc2324)"
"---
Tests implicit and explicit import. Also tests if namespace works.
!>>
!<< Hello world!
Hello world!
"

import 'std' @
('Hello world!' !println)

import 'std' \
('Hello world!' !println\Std)
```

B.51 tests/lists.t

Code Listing

```
"tests/lists.t -- Author: Zihang Chen (zc2324)"
"---
Tests various list actions.
!>>
!<< 1
2
3
[42]
[3 1 2]
[1 2]
[1 2 3 4 5]
3
"

import 'std' @

bind -my-list [1 2 3 [3 4] [42]]
then (
  -my-list #hd !println
  -my-list #tl #hd !println
  -my-list #tl #tl #tl #hd #hd "really a pain in the butt" !println
  -my-list #tl #tl #tl #tl #hd !println
)

bind -my-list [1 2]
then (
  "tests if lists are immutable"
  3 -my-list #cons !println
  -my-list !println
  -my-list [3 4 5] |#| !println
)

bind ~my-tuple [\ 1 2 [\ 3 4] [5 6] 7]
then (~my-tuple #3 #1 !println)
```

B.52 tests/macro.t

Code Listing

```
"tests/macro.t -- Author: Zihang Chen (zc2324)"
"---
Tests if macros (simulated) works.
!>>
!<< -1
"

import 'std' \

bind Macro {if>0 -\Std, +\Std}
then (1 2 Macro !println\Std)
```

B.53 tests/obj.t

Code Listing

```
"tests/obj.t -- Author: Zihang Chen (zc2324)"
"---
Some object-oriented programming sh**.
!>>
!<< 7
8
"

import 'std' @

bind ClassA fun` X,
  bind Data X
  then bind Accessor fun`, Data
       then [\Accessor]
       then bind Instance1-methods (7 ClassA)
     also Instance2-methods (8 ClassA)
     then bind Instance1-accessor (Instance1-methods #1)
          also Instance2-accessor (Instance2-methods #1)
          then (Instance1-accessor !println
        Instance2-accessor !println)
```

B.54 tests/partial.t

Code Listing

```
"tests/partial.t -- Author: Zihang Chen (zc2324)"
"---
Tests partial function application.
!>>
!<< 42
41
"

import 'std' \

bind Dec1 (1 -\Std)
then (43 Dec1 !println\Std 42 Dec1 !println\Std)
```

B.55 tests/phony.t

Code Listing

```
"tests/phony.t -- Author: Zihang Chen (zc2324)"
"---
Tests if phony works.
!>>
!<< -1
```

```
"
import 'std' @
import '.w' \

(1 $$ 2 - .!invoke\.w !println)
```

B.56 tests/pop.t

Code Listing

```
"tests/pop.t -- Author: Zihang Chen (zc2324)"
"---
Tests some of the stack instructions.
!>>
!<< 42
0
1
"
import 'std' \
import '.w' \

(42 24 !!pop\Std !println\Std)

(21 !!dup\Std -\Std !println\Std)

(21 20 2 !!pack\Std -\Std` /apply\Std !println\Std)

"(1 2 [-\Std] !println\Std)"
```

B.57 tests/q.t

Code Listing

```
"tests/q.t -- Author: Zihang Chen (zc2324)"
"---
Holistic test: Quicksort.
!>>
!<< [1 1 2 2 3 3 4 4 5 5 6 7 8 9 10]
"

import 'std' \

bind #quicksort ,\ L,
  (L ?#empty\Std ift (!!pop\Std []), (!!pop\Std
     L #tl\Std (L #hd\Std >\Std) /filter\Std #quicksort
     L #hd\Std [] #cons\Std #concat\Std
     L #tl\Std (L #hd\Std <=\Std) /filter\Std #quicksort #concat\Std))
then ([5 4 3 2 1 10 9 8 7 6 5 4 3 2 1] #quicksort !println\Std)
```

B.58 tests/quicksort.t

Code Listing

```
"tests/quicksort.t -- Author: Zihang Chen (zc2324)"
"---
Holistic test: Quicksort.
!>>
!<< [1 2 3 4 5]
"

import 'std' \

bind #quicksort ,\ L,
  (L ?#empty\Std ift (!!pop\Std []), (!!pop\Std
    bind ~h (L #hd\Std)
    also ~t (L #tl\Std)
    then (~t (~h >\Std) /filter\Std #quicksort
          [~h]
          ~t (~h <=\Std) /filter\Std #quicksort
      #concat\Std #concat\Std)))
then ([5 4 3 2 1] #quicksort !println\Std)
```

B.59 tests/rnd.t

Code Listing

```
"tests/rnd.t -- Author: Zihang Chen (zc2324)"
"---
Tests if ext module works.
!>>
!<< 0.473146049408
0.717511397385
0.11174899528
"

import 'random' @
import 'std' @

(2u ~seed
 ~~ !println
 ~~ !println
 3u ~seed
 ~~ !println)
```

B.60 tests/test.t

Code Listing

```
"tests/test.t -- Author: Zihang Chen (zc2324)"
"The Test
---
Wow! Such Hello! So World!
```

```
!>>
!<< Hello world!
"

import 'std' \

('Hello world!\n' !print\Std)
```

B.61 tests/vargs.t

Code Listing

```
"tests/vargs.t -- Author: Zihang Chen (zc2324)"
"---
Tests variadic functions.
!>>
!<< [1 2 3 4 5 4 5 6 7 8 2 3 4 5 6]
15
"

import 'std' @

$$ [1 2 3 4 5] [4 5 6 7 8] [2 3 4 5 6] #!vconcat !println

+` 0 $$ 1 2 3 4 5 /!vfoldl !println
```

B.62 tests/wscript

Code Listing

```
# tests/wscript -- Author: Zihang Chen (zc2324)
import os

def test(ctx):
    os.chdir('tests')

    ctx.exec_command(['./prefect', os.path.join('build', 'src',
                                                'compiler', 'towel'),
                      os.path.join('build', 'src',
                                   'vm', 'tvm')])
```

B.63 tests/zip.t

Code Listing

```
"tests/zip.t -- Author: Zihang Chen (zc2324)"
"---
Tests advanced enumerable literal creation.
!>>
!<< [[@ 1 2] [@ 2 3] [@ 3 4] [@ 4 5] [@ 5 6]]
"
```

```
import 'std' @

bind #zip fun` ~1 ~2,
  bind ~f ,\ ~acc ~xs1 ~xs2,
    (~xs1 ?# ift ~acc, (
     ~xs2 ?# ift ~acc, (
       [\(~xs1 #hd) (~xs2 #hd)] ~acc #cons
       ~xs1 #tl ~xs2 #tl ~f@)))
  then ([] ~1 ~2 ~f #rev@)
then ([1 2 3 4 5] [2 3 4 5 6] #zip !println)
```

B.64 `wscript`

Code Listing

```
# wscript -- Author: Zihang Chen (zc2324)
#! /usr/bin/env python3

import os
import waflib

APPNAME = 'Towel'
VERSION = '-1.-1'

top = '.'
out = 'build'

def options(opt):
    opt.add_option('--docs', action='store_true', default=False,
                   dest='compile_docs')
    opt.add_option('--compiler', action='store_true', default=False,
                   dest='compile_compiler_stub')
    opt.add_option('--native', action='store_true', default=False,
                   dest='compile_natively')
    opt.add_option('--tvm', action='store_true', default=False,
                   dest='compile_tvm')
    opt.add_option('--all', action='store_true', default=False,
                   dest='compile_all')
    opt.add_option('--debug', action='store_true', default=False,
                   dest='compile_debug')
    opt.add_option('--conf-test', action='store_true', default=False,
                   dest='conf_test')
    opt.add_option('--no-docs', action='store_true', default=False,
                   dest='no_docs')

def configure(ctx):
    ctx.options.compile_compiler = True

    if ctx.options.compile_docs:
        ctx.options.compile_compiler = False

```

```
    if ctx.options.compile_tvm:
        ctx.options.compile_compiler = False

    if ctx.options.no_docs:
        ctx.options.compile_docs = False

    if ctx.options.compile_compiler_stub:
        ctx.options.compile_compiler = True

    if ctx.options.compile_all:
       ctx.options.compile_docs = True
       ctx.options.compile_compiler = True
       ctx.options.compile_tvm = True
       ctx.options.conf_test = True

    def conf_ocaml(programs, libs):
        ctx.load('ocaml')

        ctx.find_program('ocamlfind', var='OCAMLFIND')
        for p, v in programs:
            ctx.find_program(p, var=v)

        def find_lib(l):
            ret = ctx.exec_command([ctx.env.OCAMLFIND[0], 'query', l])
            if ret:
                ctx.fatal('Cannot find library \'%s\'.' % l)
            else:
                ctx.msg('Checking for library \'%s\'' % l, 'ok')

        ctx.env.LIBS = {'Batteries', 'Extlib', 'Stdint', 'Sha'}
        ctx.env.TVM_LIBS = {'Stdint', 'Batteries', 'Extlib', 'Dynlink'}
        for l in libs:
            find_lib(l)

        ctx.env.OC = [os.path.basename(ctx.env.OCAMLC[0])]
        if ctx.options.compile_natively:
            ctx.env.OC = [os.path.basename(ctx.env.OCAMLOPT[0])]

        ctx.env.DEBUG = ''
        if ctx.options.compile_debug:
            ctx.env.DEBUG = '-g'
            ctx.env.OC = [os.path.basename(ctx.env.OCAMLC[0])]

    def conf_tex():
        ctx.load('tex')

    def conf_test():
        ctx.find_program('ruby')
        ctx.find_program('gem')

        def test_lib(what):
            ret = ctx.cmd_and_log(['gem', 'list', what],
                                  output=waflib.Context.STDOUT,
```

```
                                    quiet=waflib.Context.BOTH)
            if what in ret:
                ctx.msg('Checking for library \'%s\'' % what, 'ok')
            else:
                ctx.fatal('Cannot find library \'%s\'.' % what)

        test_lib('colorize')

    def conf_compiler():
        conf_ocaml([('ruby', 'RUBY')],
                   ['Batteries', 'Stdint', 'Extlib', 'Sha'])

    def conf_tvm():
        conf_ocaml([('ruby', 'RUBY')], ['Batteries', 'Stdint'])

    if ctx.options.compile_docs:
        conf_tex()

    if ctx.options.compile_all:
        conf_compiler()
        conf_tvm()
        conf_tex()
        conf_test()

    if ctx.options.compile_compiler:
        conf_compiler()

    if ctx.options.compile_tvm:
        conf_tvm()

    if ctx.options.conf_test:
        conf_test()

def test(ctx):
    ctx.recurse('tests')

def build(ctx):
    if ctx.options.compile_docs:
        ctx.recurse('docs')

    if ctx.options.compile_tvm:
        ctx.recurse('src/vm')

    if ctx.options.compile_compiler:
        ctx.recurse('src/compiler')

    if ctx.options.compile_all:
        ctx.recurse('src/compiler src/vm')
        ctx.recurse('docs')
```

Appendix C

References

[1] Simon Marlow. *Haskell 2010 Language Report*. https://www.haskell.org/onlinereport/haskell2010/, 2015-12-16.

[2] The GNU Project. *The GNU C Reference Manual*. http://www.gnu.org/software/gnu-c-manual/gnu-c-manual.html, 2015-12-12.

[3] Xavier Leroy, Damien Doligez, Alain Frisch, Jacques Garrigue, Didier Rémy and Jérôme Vouillon. *The OCaml System Release 4.02 - Documentation and User's Manual*. http://caml.inria.fr/pub/docs/manual-ocaml/, 2015-12-12.

[4] J.V. Noble. *A Beginner's Guide to Forth*. http://galileo.phys.virginia.edu/classes/551.jvn.fall01/primer.htm, 2015-12-12.

[5] Stephan Becher. *Introduction to StrongForth*. http://www.arestlessmind.org/2009/02/03/intro.html, 2015-12-12.

[6] Paul Bourke. *PostScript Tutorial*. http://paulbourke.net/dataformats/postscript/. 2015-12-12.

[7] *Joy (programming language)*. https://en.wikipedia.org/wiki/Joy_(programming_language). 2015-12-12.

[8] Language Perils. *The Joy of Joy*. http://ncreep.github.io/language_perils/blog/2013-03-18-the-joy-of-joy.html. 2015-12-12.

[9] Xavier Leroy, Damien Doligez, Alain Frisch, Jacques Garrigue, Didier Rémy and Jérôme Vouillon. *The OCaml Library Reference*. http://caml.inria.fr/pub/docs/manual-ocaml/libref/index.html, 2015-12-12.

[10] The OCaml Batteries Team. *Batteries User Guide*. http://ocaml-batteries-team.github.io/batteries-included/hdoc2/, 2015-12-12.

[11] Naser AlDuaij, Senyao Du, Noura Farra, Yuan Kang, Andrea Lottarini. *Funk Programming Language Final Report*. http://www.cs.columbia.edu/~sedwards/classes/2012/w4115-fall/reports/Funk.pdf, 2015-12-12.