CLIP - A Cryptal Language with Irritating Parentheses

Wei Duan (wd2214@columbia.edu)
Yi-Hsiu Chen (yc2796@columbia.edu)

Motivation

Cryptography is a discipline that has important influence on information security such as data protection, information privacy and identity authentication. It is widely applied in fields of scientific research, electronic communication, E-commerce, etc. Though highly intersected with computer science, building cryptography programs often requires much of the endeavor from programmers, as not many programming languages are able to perform even some usual cryptographic operations with convenience.

Language Features

With the issues being addressed, we decided to develop a programming language, CLIP, which emphasizes on important cryptographic operations include data shifting, bit-wise manipulation and several operations in number theory. Moreover, by allowing a rather strict memory allocation strategy, CLIP could perform calculation with higher efficiency, thus strengthen its ability of processing big numbers. From a high level viewpoint, CLIP is essentially a language that facilitates the mathematical computation of a sequence of data. Hence, we intend to implement it with a paradigm that is similar to functional languages. As most of the cryptography manipulations are pure mathematic operations, functional programming would maintain much of the true nature of cryptographic operations. In fact, CLIP shares many features with Lisp and Ocaml.

Potential Applications

CLIP is designed specifically for cryptographers. It could be adopted to express existing cryptography concepts, develop new algorithms and implement cryptographic functions with a simple and concise fashion. For example, a popular algorithm, Advanced Encryption Standard, could be implemented easily due to the specific functions, say modulo operation, provided by CLIP. In addition, CLIP has built-in support for data shifting, data rotation, type casting, logic operations like xor, etc. All of them help to reduce the level of complication in cryptographic programing.

Language Specification

1. Data types

Key Word	Description
int#n	n bit integer
char	single character
poly	integer-valued polynomial
vector	a sequence of integers, characters, polynomials, strings or vectors
string	a sequence of characters

2. Built-in Function

Other than some common functions and operator in programming language, including '+', '-', '*', '%', 'and', 'or', 'not', 'xor' and relational function, CLIP also have below built-in function.

Basic functions

Function name	Description
defun	Defining a function
defvar	Defining a global variable
let	Binding of a variable or a function
map	Applying a function to all elements in a vector
cast	Casting data type

Vector function

Function name	Description
car	The first element in a vector
cons	Inserting an element to the begining of a vector
cdr	A vector with the first element removed
concat	Concatinating vectors into one large vector
transpose	Transposing a vector

Operators for shifting and rotation

Function name	Description
<<	Left shifting
>>	Right shifting
<<<	Left rotating
>>>	Right rotating

Miscellaneous function

Function name	Description
inv	Inverse of multiplication
pow	Exponentiation
gcd	Finding the greatest common divisor
group	Dividing a vector into a group of subvectors
merge	Combining several small vectors into a vector with subvectors
len	Number of elements in a vector or number of bits of an integer
reverse	Reversing a vector or the sequence of bit of an integer

3. Syntax

comments

```
~ This is a single line comment
~~~ This is a
    long comment. ~~~
```

function definition

```
defun f para1 para2 ... =
    do-something;
```

function call

```
(function-name para1 para2 ...)
```

control flow

```
if expression then
    do-something
else if
    do-something
else
    do-something
```

local assignments

```
(let <var1 value1>
     <var2 value2>
     ......
     do-something
)
```

Sample Program

```
~~~ global variable binding ~~~
defvar pi 3.1415;
~~~ function definition ~~~
defun lcm a:int# b:int# =
    (* (gcd a b) a b);
~~~ polynomial calculation ~~~
(let <a $1 5x^3 9x^5$#4>
     <b $3 4x^3 7x^4 10x^5$#4>
     (+ a b))
=> $4 9x^3 7x^4 3x^5$
~~~ some vector operations ~~~
(concat [1, 2, 3] [4, 5, 6]);
=> [1, 2, 3, 4, 5, 6]
(split 2 [1, 2, 3, 4, 5, 6]);
=> [[1, 2], [3, 4], [5, 6]]
~~~ Fibonacci number ~~~
defun fib x:int# =
    if (= x 0) then
        0
    else if (= x 1) then
        1
    else
        (+ (fib (1- x)) ((- x 2)));
```