Picrin Documentation

Release 0.1

Yuichi Nishiwaki and other picrin contributors

Jun 11, 2017

Contents

Contents:

CHAPTER 1

Introduction

Picrin is a lightweight R7RS scheme implementation written in pure C89. It contains a reasonably fast VM, an improved hygienic macro system, usuful contribution libraries, and simple but powerful C interface.

- R7RS compatible
- Reentrant design (all VM states are stored in single global state object)
- Bytecode interpreter
- Direct threaded VM
- Internal representation by nan-boxing (available only on x64)
- Conservative call/cc implementation (VM stack and native c stack can interleave)
- Exact GC (simple mark and sweep, partially reference count)
- String representation by rope
- Hygienic macro transformers (syntactic closures, explicit and implicit renaming macros)
- Extended library syntax

Homepage

Currently picrin is hosted on Github. You can freely send a bug report or pull-request, and fork the repository.

https://github.com/picrin-scheme/picrin

Documentation

See http://picrin.readthedocs.org/

IRC

There is a chat room on chat.freenode.org, channel #picrin. IRC logs here: https://botbot.me/freenode/picrin/

LICENSE

CHAPTER 2

Installation

Installation instructions below.

Build

Just type *make* in the project root directory. You will find an executable binary newly created at bin/ directory.

```
$ make
```

When you are building picrin on x86_64 system, PIC_NAN_BOXING flag is automatically turned on (see include/picrin/config.h for detail).

Install

make install target is provided. By default it installs picrin binary into */usr/local/bin/*.

```
$ make install
```

Since picrin does not use autoconf, if you want to specify the install directory, pass the custom path to *make* via command line argument.

```
$ make install prefix=/path/to/dir
```

Requirement

To build Picrin Scheme from source code, some external libraries are required:

- perl
- regex.h of POSIX.1
- libedit (optional)

Make command automatically turns on optional libraries if available. Picrin is mainly developed on Mac OS X and only tested on OS X or Ubuntu 14.04+. When you tried to run picrin on other platforms and found something was wrong with it, please send us an issue.

CHAPTER 3

Language

Picrin's core language is the R7RS scheme with some powerful extensions. Please visit http://r7rs.org/ for the information of R7RS's design and underlying thoughts.

The REPL

At the REPL start-up time, some usuful built-in libraries listed below will be automatically imported.

- `(scheme base)`
- `(scheme load)`
- `(scheme process-context)`
- `(scheme write)`
- `(scheme file)`
- `(scheme inexact)`
- `(scheme cxr)`
- `(scheme lazy)`
- `(scheme time)`
- `(scheme case-lambda)`
- `(scheme read)`
- `(scheme eval)`

Compliance with R7RS

section	status	comments
2.2 Whitespace and comments	yes	
2.3 Other notations	incomplete	#e #i #b #o #d #x
2.4 Datum labels	yes	
3.1 Variables, syntactic keywords, and regions		
3.2 Disjointness of types	yes	
3.3 External representations		
3.4 Storage model	yes	
3.5 Proper tail recursion	yes	As the report specifies, `apply`, `call/cc`, and `call-with-val`
4.1.1 Variable references	yes	
4.1.2 Literal expressions	yes	
4.1.3 Procedure calls	yes	In picrin `()` is self-evaluating
4.1.4 Procedures	yes	
4.1.5 Conditionals	yes	In picrin `(if #f #f)` returns `#f`
4.1.6 Assignments	yes	
4.1.7 Inclusion	incomplete	`include-ci`
4.2.1 Conditionals	yes	
4.2.2 Binding constructs	yes	
4.2.3 Sequencing	yes	
4.2.4 Iteration	yes	
4.2.5 Delayed evaluation	yes	
4.2.6 Dynamic bindings	yes	
4.2.7 Exception handling	yes	`guard` syntax.
4.2.8 Quasiquotation	yes	can be safely nested. TODO: multiple argument for unquote
4.2.9 Case-lambda	yes	
4.3.1 Bindings constructs for syntactic keywords	yes[1]	
4.3.2 Pattern language	yes	`syntax-rules`
4.3.3 Signaling errors in macro transformers	yes	
5.1 Programs	yes	
5.2 Import declarations	yes	
5.3.1 Top level definitions	yes	
5.3.2 Internal definitions	yes	
5.3.3 Multiple-value definitions	yes	
5.4 Syntax definitions	yes	
5.5 Recored-type definitions	yes	
5.6.1 Library Syntax	yes	In picrin, libraries can be reopend and can be nested.
5.6.2 Library example	N/A	
5.7 The REPL	yes	
6.1 Equivalence predicates	yes	
6.2.1 Numerical types	yes	picrin has only two types of internal representation of numbers: fixn
6.2.2 Exactness	yes	
6.2.3 Implementation restrictions	yes	
6.2.4 Implementation extensions	yes	
6.2.5 Syntax of numerical constants	yes	
6.2.6 Numerical operations	yes	`denominator`, `numerator`, and `rationalize` are not suppor
6.2.7 Numerical input and output	yes	
6.3 Booleans	yes	
6.4 Pairs and lists	yes	`list?` is safe for using against circular list.
6.5 Symbols	yes	
6.6 Characters	yes	
6.7 Strings	yes	

Table 3.1 – continued from previous page

section	status	comments
6.8 Vectors	yes	
6.9 Bytevectors	yes	
6.10 Control features	yes	
6.11 Exceptions	yes	
6.12 Environments and evaluation	yes	
6.13.1 Ports	yes	
6.13.2 Input	yes	
6.13.3 Output	yes	
6.14 System interface	yes	

[1] Picrin provides hygienic macros in addition to so-called legacy macro (`define-macro`), such as syntactic closure, explicit renaming macro, and implicit renaming macro.

CHAPTER 4

Standard Libraries

Picrin's all built-in libraries are described below.

(picrin macro)

Utility functions and syntaces for macro definition.

- define-macro
- gensym
- ungensym
- macroexpand
- macroexpand-1

Old-fashioned macro.

- identifier?
- identifier=?
- make-syntactic-closure
- close-syntax
- capture-syntactic-environment
- sc-macro-transformer
- rsc-macro-transformer

Syntactic closures.

- er-macro-transformer
- ir-macro-transformer
- strip-syntax

Explicit renaming macro family.

(picrin array)

Resizable random-access list.

Technically, picrin's array is implemented as a ring-buffer, effective double-ended queue data structure (deque) that can operate pushing and poping from both of front and back in constant time. In addition to the deque interface, array provides standard sequence interface similar to functions specified by R7RS.

- **(make-array [capacity])**

  Returns a newly allocated array object. If capacity is given, internal data chunk of the array object will be initialized by capacity size.

- **(array . objs)**

  Returns an array initialized with objs.

- **(array? . obj)**

  Returns #t if obj is an array.

- **(array-length ary)**

  Returns the length of ary.

- **(array-ref ary i)**

  Like `list-ref`, return the object pointed by the index i.

- **(array-set! ary i obj)**

  Like `list-set!`, substitutes the object pointed by the index i with given obj.

- **(array-push! ary obj)**

  Adds obj to the end of ary.

- **(array-pop! ary)**

  Removes the last element of ary, and returns it.

- **(array-unshift! ary obj)**

  Adds obj to the front of ary.

- **(array-shift! ary)**

  Removes the first element of ary, and returns it.

- **(array-map proc ary)**

  Performs mapping operation on ary.

- **(array-for-each proc ary)**

  Performs mapping operation on ary, but discards the result.

- **(array->list ary)**

  Converts ary into list.

- **(list->array list)**

  Converts list into array.

(picrin dictionary)

Symbol-to-object hash table.

- **(make-dictionary)**

  Returns a newly allocated empty dictionary.

- **(dictionary . plist)**

  Returns a dictionary initialized with the content of plist.

- **(dictionary? obj)**

  Returns #t if obj is a dictionary.

- **(dictionary-ref dict key)**

  Look up dictionary dict for a value associated with key. If dict has a slot for key *key*, a pair containing the key object and the associated value is returned. Otherwise *#f* is returned.

- **(dictionary-set! dict key obj)**

  If there is no value already associated with key, this function newly creates a binding of key with obj. Otherwise, updates the existing binding with given obj.

  If obj is *#undefined*, this procedure behaves like a deleter: it will remove the key/value slot with the name *key* from the dictionary. When no slot is associated with *key*, it will do nothing.

- **(dictionary-size dict)**

  Returns the number of registered elements in dict.

- **(dicitonary-map proc dict)**

  Perform mapping action onto dictionary object. `proc` is called by a sequence `(proc key1 key2 ...)`.

- **(dictionary-for-each proc dict)**

  Similar to `dictionary-map`, but discards the result.

- **(dictionary->plist dict)**
- **(plist->dictionary plist)**
- **(dictionary->alist dict)**
- **(alist->dictionary alist)**

  Conversion between dictionary and alist/plist.

(picrin user)

When you start the REPL, you are dropped into here.

CHAPTER 5

C API

You can write Picrin's extension by yourself from both sides of C and Scheme. This page describes the way to control the interpreter from the C world.

Extension Library

If you want to create a contribution library with C, the only thing you need to do is make a directory under contrib/. Below is a sample code of extension library.

- contrib/add/nitro.mk

```
CONTRIB_INITS += add
CONTRIB_SRCS  += contrib/add/add.c
```

- contrib/add/add.c

```
#include "picrin.h"

static pic_value
pic_add(pic_state *pic)
{
  double a, b;

  pic_get_args(pic, "ff", &a, &b);

  return pic_float_value(pic, a + b);
}

void
pic_init_add(pic_state *pic)
{
  pic_deflibrary (pic, "(picrin add)") {
    pic_defun(pic, "add", pic_add);
```

```
  }
}
```

After recompiling the interpreter, the library "(picrin add)" is available in the REPL, which library provides a funciton "add".

User-data vs GC

When you use dynamic memory allocation inside C APIs, you must be caseful about Picrin's GC. Fortunately, we provides a set of wrapper functions for complete abstraction of GC. In the case below, the memory (de)allocators *create_foo* and *finalize_foo* are wrapped in pic_data object, so that when an instance of foo losts all references from others to it picrin can automatically finalize the orphan object.

```
/** foo.c **/
#include <stdlib.h>
#include "picrin.h"

/*
 * C-side API
 */

struct foo {
  // blah blah blah
};

struct foo *
create_foo ()
{
  return malloc(sizeof(struct foo));
}

void
finalize_foo (void *foo) {
  struct foo *f = foo;
  free(f);
}


/*
 * picrin-side FFI interface
 */

static const pic_data_type foo_type = { "foo", finalize_foo };

static pic_value
pic_create_foo(pic_state *pic)
{
  struct foo *f;

  pic_get_args(pic, ""); // no args here

  f = create_foo();

  return pic_data_value(pic, md, &foo_type);
}
```

```
void
pic_init_foo(pic_state *pic)
{
  pic_defun(pic, "create-foo", pic_create_foo); // (create-foo)
}
```

CHAPTER 6

Indices and tables

- genindex
- modindex
- search