Languages Implementation with Parrot
A study case: Lua on Parrot

François Perrad
francois.perrad@gadz.org
PAUSE ID: PERRAD
Parrot SVN: fperrad

Languages in Parrot

- today, more than 50
  - see languages/LANGUAGES_STATUS.pod
- PIR generators (antlr, yapp, yacc, …)
  - jako, basic, PJS
- PIR generators written with PIR
  - TCL, Python, Ruby, Perl, PHP, Lua
- Bytecode Translators
  - dotnet, WMLScript
- Obfuscated
  - bf, befunge, ook

Parrot Compiler Toolkit

- Patrick R. Michaud
  - See his presentation at YAPC::NA2007
- Parser Grammar Engine
- Tree Grammar Engine
- Parrot Abstract Syntax Tree
  - *Node*, Op, Val, Var, Stmts, Block
- Parrot Opcode Syntax Tree
  - *Node*, Ops, Op, Label, Sub
- HLLCompiler

Why choose Lua

- Prove the design goal :
  « supports all dynamic languages »
- Lua has advanced features
- Lua is small & simple

- A good way to learn Lua

Lua

- Academic origin (1993, Brazil)
- Open Source
  - but Closed Development
- Often embedded into applications
- Widely used in Game industry
- www.lua.org

Lua features

- function as first class
- exception (as library)
- closure
- coroutine (as library)
- iterator
- regex (with its own dialect)
- some OO mechanisms
- tail call

Lua is small

- 8 types
  - nil, boolean, number, string, function, userdata, thread, table
- 21 keywords
  - No builtin function,
  - only standard libraries
- Grammar EBNF
  - 1 page
- Reference Manual
  - < 100 pages

Lua On Parrot

- Native PMC
- Tests
- Compiler based on Parse::Yapp
- Lua Standard Libraries
- Interpreter based on PGE/TGE

Lua PMC

- Native implementation of all Lua types
  - As a shared library or a DLL
- Preprocessed C dialect
  - VTABLE methods
  - Non-VTABLE methods
  - Specifics methods
- Allows in PIR :

```
.HLL 'Lua', 'lua_group'

.local pmc MyStr
new MyStr, 'LuaString'
set MyStr, 'Some text'
```

Lua PMC

```
pmclass LuaNumber extends LuaAny does float
        dynpmc group lua_group hll Lua maps Float {

    void init() {
        PMC_num_val(SELF) = 0.0;
    }

    STRING* name() {
        return const_string(INTERP, "number");
    }

    void increment() {
        PMC_num_val(SELF) ++;
    }
```

Lua PMC

```
    INTVAL cmp(PMC* value) {
MMD_LuaNumber: {
        FLOATVAL diff;
        diff = PMC_num_val(SELF) - PMC_num_val(value);
        return diff > 0 ? 1 : diff < 0 ? -1 : 0;
    }
MMD_DEFAULT: {
        real_exception(INTERP, NULL, ILL_INHERIT,
            "attempt to compare number with %s",
             string_to_cstring(INTERP, VTABLE_name(INTERP, value)));
            return 0;
        }
    }
```

Lua Tests

- as part of smoke test
  - make languages-smoke
  - http://smoke.parrotcode.org/smoke/
- a framework from Perl
  - use Test::More;
  - pir_output_is, pir_output_like, …
  - language_output_is, language_output_like
- Test Driven Development
- today, Lua has more than 970 tests

Lua Tests

```
# file: examples.t
 …
language_output_is('lua',<<'CODE',<<'OUT','factorial');
function factorial (n)
    if n == 0 then
        return 1
    else
        return n * factorial(n-1)
    end
end

print(factorial(7))
CODE
5040
OUT
```

luac.pl with Parse::Yapp

- PGE was not available at this time
- Yapp is a good tool
- Split complexity
  - What PIR needed by Lua
  - How generate it

Lua Standard Libraries

- 9 libraries written in PIR:
  - basic (24/25), coroutine (6/6), package (3/4), string (13/14), table (8/8), math (28/28), IO (17/18), OS (11/11), debug (7/14)
- More complex functions are :
  - Eval,
  - Exception,
  - Regex (extends PGE)
- A huge work

Lua Standard Libraries

```
=item C<string.len (s)>

Receives a string and returns its length.

=cut

.sub 'len' :anon
    .param pmc s :optional
    .param pmc extra :slurpy
    .local pmc res
    $S1 = lua_checkstring(1, s)
    $I0 = length $S1
    new res, 'LuaNumber'
    set res, $I0
    .return (res)
.end
```

Lua on Parrot

- PGE
  - Lua grammar + optable : src/lua51.pg
- TGE
  - PAST-pm : src/PASTGrammar.tg
  - POST : src/POSTGrammar.tg
- HLLCompiler
  - Lua::Compiler + Utils : src/lua51.pir
- Standalone interpreter
  - Entry point : lua.pir
  - A single PBC : lua.pbc
- PIR generation & execution

Lua on Parrot

- A lexicography test : test_lex.pir
- luap.pir –target=PARSE|PAST|POST|PIR

```
.sub 'main' :main
    .param pmc args
    load_bytecode 'languages/lua/lua.pbc'
    $P0 = compreg 'Lua'
    $S0 = "Compiler Lua 5.1 on Parrot"
    $P0.commandline_banner($S0)
    $P0.command_line(args)
.end
```

Lua – Parse

```
"parse" => PMC 'Lua::Grammar' => "print \"Hello world!\"\r\n" @ 0 {
  <block> => PMC 'Lua::Grammar' => "print \"Hello world!\"\r\n" @ 0 {
    <statement> => ResizablePMCArray (size:1) [
      PMC 'Lua::Grammar' => "print \"Hello world!\"\r\n" @ 0 {
        <expression_stat> => PMC 'Lua::Grammar' => "print \"Hello world!\"\r\n" @ 0
          <primary_expression> => PMC 'Lua::Grammar' => "print \"Hello world!\"\r\n
            <prefix_expression> => PMC 'Lua::Grammar' => "print " @ 0 {
              <Name> => PMC 'Lua::Grammar' => "print" @ 0
            }
            <slice_expression> => ResizablePMCArray (size:1) [
              PMC 'Lua::Grammar' => "\"Hello world!\"\r\n" @ 6 {
                <function_args> => PMC 'Lua::Grammar' => "\"Hello world!\"\r\n" @ 6
                  <string> => PMC 'Lua::Grammar' => "\"Hello world!\"" @ 6 {
                    <quoted_literal> => PMC 'Lua::Grammar' => "Hello world!" @ 7
                  }
                }
```

Lua – PAST

```
"past" => PMC 'PAST::Block'  {
    [0] => PMC 'PAST::Var'  {
        <name> => "vararg"
        <scope> => "parameter"
        <isslurpy> => 1
    }
    [1] => PMC 'PAST::Stmts'  {
        [0] => PMC 'PAST::Op'  {
            <pasttype> => "call"
            [0] => PMC 'PAST::Var'  {
                <name> => "print"
                <scope> => "package"
            }
            [1] => PMC 'PAST::Val'  {
                <vtype> => "LuaString"
                <name> => "Hello world!"
            }
        }
```

Lua - PIR

```
.HLL "Lua", "lua_group"

.sub "&start"  :anon :main
    …
.end

.sub "&main_10" :outer("&start") :anon :lex
    .param pmc vararg    :slurpy
    .const .LuaString k_print = "print"
    .local pmc subr
    subr = interpinfo .INTERPINFO_CURRENT_SUB
    $P11 = subr.getfenv()
    set $P12, $P11[k_print]
    new $P13, "LuaString"
    set $P13, "Hello world!"
    $P12($P13)
.end
```

Status of Lua on Parrot

- Garbage Collection problems
  - performance
- Parrot IO & OS subsystems are incomplete
- Next step : shift to PCT

- See languages/lua/doc/status.pod

Feedback to Parrot

- Lua on Parrot is successful
- Tool effectiveness
  - Only 4 kLoC
- Learning curve challenge

Back to Perl6 / Parrot

- Conditions of success
  - Language Perl6 : **OK**
  - Parrot Core VM : **OK**
  - Parrot Compiler Toolkit : **OK**
  - Wrapper Toolkit
    - NCI is better than Perl XS
    - The state of the art : Python pyrex & ctype
  - Distributed Computing
    - ~ Java RMI