URCC: Rochester/Ruby C Compiler

Chen Ding
Lingxiang Xiang

First release: repos v256, March 17, 2014
Discussion board:

```
[repos]/assignments/test_cases $  ruby ../../urcc/bin/trivial_pass ab.c

Generating unoptimized binary 'ab_urcc.bin'

 gcc -O0 -Wno-format-security -Wno-implicit-function-declaration -g ab_urcc.c -o
ab_urcc.bin

Invoking trivial pass

Generating optimized binary 'ab_urcc_opt.bin'

 gcc -O0 -Wno-format-security -Wno-implicit-function-declaration -g ab_urcc_opt.c -o
ab_urcc_opt.bin
```

Files generated in [repos]/assignments/test_cases/

```
ab_urcc.c                    .urcc_log etc
ab_urcc.bin
ab_urcc_opt.c
ab_urcc_opt.bin
```

(Do not check in generated files)

```
#include <stdio.h>

int main(){

  int x, y, z;
  y = 0;
  x = 1;
  z = x + y;
  printf ("19 s=%2d\n", z);

}
```

ab.c

```
#include <stdio.h>
#include <stdlib.h>

  /* Ast::Scope prog: 1 symbols, 1
children */
  char str[10] = "19 s=%2d\n";

int main (){
    /* Ast::Block main body: 9 symbols, 11
children */
    int var_x;
    int var_y;
    int var_z;
    int var_reg2mem_alloca_point;
    int var_1;
    int var_2;
    int var_3;
    int var_4;
    int var_5;

    printf("function main called\n");
    var_reg2mem_alloca_point = 0;
    var_y = 0;
    var_x = 1;
    var_1 = var_x;
```

ab_urcc_opt.c

```
#!/usr/bin/ruby

$Passes = $Passes || Array.new

# insert "printf(function_name)" at the beginning of every function
trivial_pass = Proc.new do |prog|
  # only one module in a program
  prog.each { |func|
    # skip function decl
    next if not func.is_a?Ast::Func or func.body == nil
    call_expr = Ast::Call.new("printf")
    call_expr.add_param(Ast::StrConst.new("function #{func.id} called\\n"))
    call_stmt = Ast::AssignStat.new(call_expr)
    if func.body.size > 0
      call_stmt.insert_me("before", func.body.child(0))
    else
      call_stmt.insert_me("childof", func.body)
    end
  }
end

$Passes << ["trivial", trivial_pass]

load 'urcc'
```

```
[repos]/urcc/bin/urcc:
... ...
#########################################################
  # Optimizations are here for "prog"

  if $Passes != nil
    $Passes.each do |opt|
      puts "Invoking #{opt[0]} pass"
      opt[1].call( prog )
    end
  else
    puts 'no optimization specified'
  end

  #########################################################

  opt_file = filename[0..-3]+"_urcc_opt.c"
  file = File.new(opt_file, "w")
  file << URCCFE.dump_prog(prog)
  file.close
```

```
# Logfile created on 2014-03-19 10:50:40 -0400 by logger.rb/36483
I, [2014-03-19T10:50:40.085936 #49380]  INFO -- :  SeqRule created #<SeqRule:0x007fc6012
I, [2014-03-19T10:50:40.086061 #49380]  INFO -- :  RepRule created #<RepRule:0x007fc6012
... ...

I, [2014-03-19T10:50:40.089635 #49380]  INFO -- : #<AltRule:0x007fc601211820> looks ahead
chooses #<Literal:0x007fc601211af0>
```

.urcc_log

URCC Class Hierarchy

Mo Zi (Motsu) 480 BC

- Three types of names for everything
  - 达
    - the set of everything, the universe
  - 类
    - a set, e.g. people
  - 私
    - one particular member, e.g. Mo Zi
- Whorf-Sapir hypothesis
  - "What you could and did think was determined by the language you spoke and that some languages allowed you to think "better" than others."

Object-oriented Systems

- Statistics of a typical OO project
  - 100 classes per application
  - 1000 to model an entire domain, e.g. business enterprise
  - 3 development iterations
  - 25 to 30 percent code discarded each iteration
  - 12 methods per class
  - 15 lines of code per method in C++ or Java
- "Failing to achieve these metrics is a clear sign of the absence of object thinking"

 source: *Microsoft Object Thinking* by David West

Class Hierarchy

Ast module

- **Ast::Node**: can have one parent and any number of chidren
  - **Ast::Scope**: representing program and file, with symbol table
    - **Ast::Block**: code block and function body
    - **Ast::Func**: function
  - **Ast::Stat**: statement, i.e. anything that ends with `;`
    - **Ast::AssignStat**: assignment or just an expression (no LHS, e.g. a function call)
    - **Ast::GotoStat**: go-to or conditional go-to
    - **Ast::LabelStat**: target of go-to
    - **Ast::ReturnStat**: return expression
  - **Ast::Expr**: expression
    - **Ast::OpExpr**: unary or binary
    - **Ast::Call**: function call
    - **Ast::VarAcc**: variable access
    - **Ast::Const**: divided into **Ast::NumConst** and **Ast::StrConst**

Decl module

- **Decl::Type**: base type, with any number of indirection
  - **Decl::PrimType**: primitive types
  - **Decl::StructType**: to be implemented
  - **Decl::ArrayType**:
  - **Decl::FuncType**:
- **Decl::Var**: variable declaration, name plus Decl::Type

Traversing Objects

- Arrays
  - e.g. a = [1, 2, 3]
  - iterator
    - is it the same as passing a function pointer?
- Recursive data type
  - e.g. a program tree
  - iterator
- Do not write loops

Ast::Node Iterator

```
module Ast

class Node
  attr_reader :id, :parent
  # uncomment this for debugging
  # attr_reader :children
  ... ...
  # traverse itself and every child in the sub-tree
  def each(order="preorder", &visitor)
    raise "Unknown order\n" unless order=="preorder" or order=="postorder"
    visitor.call(self) if order == "preorder"
    @children.each do |child|
      child.each(order, &visitor)
    end
    visitor.call(self) if order == "postorder"
  end
```

each_with_level

```
# traverse in pre-order but with level info
def each_with_level(level=0, &visitor)
  raise "Be positive (or at least equivocal)\n" if level<0
  visitor.call(level, self)
  @children.each do |child| child.each_with_level(level+1, &visitor) end
  return nil
end
```


Program Transformation

- Constructors
  - program, functions, statements, expressions, constants, declarations
    - 22 Ast and Decl classes
- Symbol table, i.e. Scope::add_sym
- Ast tree construction, i.e. Ast::add_child
- Consistency check
  - Ast::sym_consistent?
    - [repos]/urcc/ast/ast_tree.rb
- Code generation
  - URCCFE.dump_prog(prog)
- Example: [repos]/urcc/bin/{trivial_pass.rb, urcc.rb}

Open-Closed Principle

Software entities (classes, modules, functions, etc.) should be open for extension, but closed for modification.

Bertrand Meyer

AST Annotation by Mix-in

Hash Tag Mixin

```
module Tags
    attr_reader :tags

    def initialize
      @tags = Hash.new
    end
end
```

```
class String
  include Tags
end

s = String.new('...')
s.tags['author'] = 'dickens'
s.tags
=>  #{'author' -> 'dickens'}
```


Set Taggable Mixin (obsolete coding style)

```
module Taggable
    attr_accessor :tags

    def taggable_setup
      @tags = Set.new
    end

    def add_tag(tag)
      @tags << tag
    end

    def remove_tag(tag)
      @tags.delete(tag)
    end
end
```

```
class TaggableString < String
  include Taggable
  def initialize(*args)
    super
    taggable_setup
  end
end
s = TaggableString.new('...')
s.add_tag 'dickens'
s.add_tag 'quotation'
s.tags # =>
#<Set: {"dickens", "quotation"}>
```

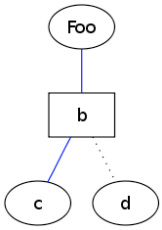
source: ***Ruby Cookbook*** by Carlson and Richardson
safaribookonline.com

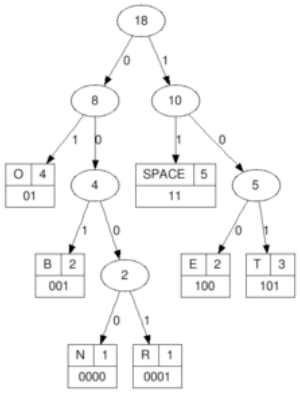
Ruby Graph Library

- http://rgl.rubyforge.org/rgl/index.html
  - "much influenced" by the C++ Boost Graph Library (BGL)
- Ruby
  - mixins (not as efficient as templates but "readable")
  - iterators (or an emumerator called Stream)
  - attributes stored in Hash

GraphViz

- Graph types
  - directed
  - undirected
  - tree
- Attributes





Summary

- Write an optimizing compiler
  - "We write the software that writes the software"
- Software design
  - object orientation
    - no big classes and giant functions
  - do not use loops
    - iterators only (ask if you need a new one)
  - modularize (mix-ins)
  - code reuse
    - leverage existing libraries
- Learn together
  - share questions, answers, ideas, tools, designs

Friday's Help Session

- Graph theoretic approaches to program analysis
  - Reps paper
  - Yannakakis paper (below)
  - Harris seminar on secure programming
- reading: "Graph-theoretic methods in database theory" by Mihalis Yannakakis, April 1990, PODS '90: Proceedings of the ninth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems