Brian Cullinan

Project 7: Advanced Analyzer

10/29/08

Objectives:

Create an advanced language analyzer that can collect, identify, validate, and transform the parse tree.

Overview:

I created analyzers, and Info types for handling the parse tree. As suggested there is a VarInfo object, a FunctionInfo, and a TypeInfo. I used 2 different types of analyzers, the main analyzer and a collection analyzer. The collection analyzer simply collects all the data that the main analyzer will need. The lists of Info is transferred to the main analyzer when the collection analyzer is complete.

Approach:

My approach to this problem was to create a class that collects the data, and a separate part of the analyzer that processes it. I have used this approach before in web applications and it turns out well. Data collection is kept separate from how it is used. My base analyzer creates a subclass of analyzer that simply collects information. It then analyzes and does the rest of the work itself. The collector returns all the lists of stuff that it collects. The main Analyzer uses this information to identify, validate, and transform. Some transformations are handled by the parser already. For example my parser can recognize numeric types without the explicit type definition in front of the value. Other transformations include automatically declaring an assignment within a ForLoop. My expression type evaluation works like this; the types are arranged in order of overriding. For example, if an integer is added to a string, the result would be a string, so a string would override the integer type. This functionality can be seen under leaveExpression(). My visit methods are somewhat unique, it is very similar to event based stream reading. This would eventually give me the option of making the entire compiler stream readable. When a node in the tree is hit, it fires another function, when it is closed it fires a leaveNode function.

Conclusion:

The analyzer turned out well. Everything seems to be in order and ready for code generation. I plan to create a virtual machine in PHP to accomplish the execution of the code immediately.

I used a var\_dump() because the output and whitespace is more compressed:

TypeInfo:

array(5) {

[0]=>

&object(TypeInfo)#192 (3) {

["name"]=>

string(13) "BinaryLiteral"

["index"]=>

int(0)

["size"]=>

int(1)

}

[1]=>

&object(TypeInfo)#193 (3) {

["name"]=>

string(10) "IntLiteral"

["index"]=>

int(1)

["size"]=>

int(8)

}

[2]=>

&object(TypeInfo)#194 (3) {

["name"]=>

string(11) "RealLiteral"

["index"]=>

int(2)

["size"]=>

int(16)

}

[3]=>

object(TypeInfo)#195 (3) {

["name"]=>

string(10) "HexLiteral"

["index"]=>

int(3)

["size"]=>

int(8)

}

[4]=>

object(TypeInfo)#196 (3) {

["name"]=>

string(13) "StringLiteral"

["index"]=>

int(4)

["size"]=>

int(8)

}

}

VarInfo:

array(5) {

[0]=>

object(VarInfo)#198 (5) {

["level"]=>

int(1)

["token"]=>

object(Token)#6 (5) {

["file"]=>

string(8) "code.txt"

["line"]=>

int(2)

["col"]=>

int(1)

["type"]=>

int(0)

["content"]=>

string(1) "v"

}

["index"]=>

int(0)

["name"]=>

string(4) "fib1"

["type"]=>

object(TypeInfo)#193 (3) {

["name"]=>

string(10) "IntLiteral"

["index"]=>

int(1)

["size"]=>

int(8)

}

}

[1]=>

object(VarInfo)#199 (5) {

["level"]=>

int(1)

["token"]=>

object(Token)#15 (5) {

["file"]=>

string(8) "code.txt"

["line"]=>

int(3)

["col"]=>

int(1)

["type"]=>

int(0)

["content"]=>

string(1) "v"

}

["index"]=>

int(1)

["name"]=>

string(4) "fib2"

["type"]=>

object(TypeInfo)#193 (3) {

["name"]=>

string(10) "IntLiteral"

["index"]=>

int(1)

["size"]=>

int(8)

}

}

[2]=>

object(VarInfo)#200 (5) {

["level"]=>

int(1)

["token"]=>

object(Token)#36 (5) {

["file"]=>

string(8) "code.txt"

["line"]=>

int(6)

["col"]=>

int(31)

["type"]=>

int(6)

["content"]=>

string(2) "

"

}

["index"]=>

int(2)

["name"]=>

string(1) "i"

["type"]=>

object(TypeInfo)#193 (3) {

["name"]=>

string(10) "IntLiteral"

["index"]=>

int(1)

["size"]=>

int(8)

}

}

[3]=>

object(VarInfo)#201 (5) {

["level"]=>

int(2)

["token"]=>

object(Token)#51 (5) {

["file"]=>

string(8) "code.txt"

["line"]=>

int(10)

["col"]=>

int(8)

["type"]=>

int(0)

["content"]=>

string(1) "l"

}

["index"]=>

int(3)

["name"]=>

string(6) "newfib"

["type"]=>

object(TypeInfo)#192 (3) {

["name"]=>

string(13) "BinaryLiteral"

["index"]=>

int(0)

["size"]=>

int(1)

}

}

[4]=>

object(VarInfo)#203 (4) {

["level"]=>

int(1)

["token"]=>

array(0) {

}

["index"]=>

int(4)

["name"]=>

string(4) "pvar"

}

}

FunctionInfo:

array(2) {

[0]=>

object(FunctionInfo)#197 (5) {

["level"]=>

int(0)

["name"]=>

string(1) "p"

["index"]=>

int(0)

["parameters"]=>

int(1)

["body"]=>

array(0) {

}

}

[1]=>

object(FunctionInfo)#202 (5) {

["level"]=>

int(1)

["name"]=>

string(5) "print"

["index"]=>

int(1)

["parameters"]=>

int(1)

["body"]=>

object(CommandList)#186 (2) {

["token"]=>

object(Token)#93 (5) {

["file"]=>

string(8) "code.txt"

["line"]=>

int(22)

["col"]=>

int(1)

["type"]=>

int(8)

["content"]=>

string(1) "]"

}

["tree"]=> ... See Below

}

Because of the method of listing out the tree structure, the parse tree is 56 pages which is too long to print. You can view it on the site for now, and I will send it in an e-mail.

<http://209.250.30.30/compiler/>

Visitor.php (This is really the only class that had big changes):

<?php

require\_once('nodes.php');

require\_once('error.php');

class Visitor

{

var $tree;

function Visitor($tree)

{

$this->tree = $tree;

}

function visit($tree)

{

if(is\_object($tree))

{

$f\_name = 'visit' . get\_class($tree);

if(method\_exists($this, $f\_name))

call\_user\_func(array($this, $f\_name), $tree);

else

$this->visitNode($tree);

$this->visit($tree->tree);

if(method\_exists($this, 'leave' . get\_class($tree)))

call\_user\_func(array($this, 'leave' . get\_class($tree)), $tree);

else

$this->leaveNode($tree);

}

else

{

if(is\_array($tree))

{

foreach($tree as $key => $object)

{

$this->visit($object);

}

}

}

}

function visitNode($tree)

{

}

function leaveNode($tree)

{

}

}

class Analyzer extends Visitor

{

var $typetable = array();

var $functable = array();

var $vartable = array();

var $reverse\_vartable = array();

function analyze()

{

$collector = new Collector($this->tree);

$collector->analyze();

$this->typetable = $collector->typetable;

$this->functable = $collector->functable;

$this->vartable = $collector->vartable;

$this->reverse\_vartable = $collector->reverse\_vartable;

$this->visit($this->tree);

var\_dump($this->typetable);

var\_dump($this->vartable);

var\_dump($this->functable);

var\_dump($this->tree);

}

var $levelcounter = 0;

function visitCommandList($tree)

{

$this->levelcounter++;

}

function leaveCommandList($tree)

{

$this->levelcounter--;

}

function getVar($level, $name)

{

$output = array();

foreach($this->vartable as $index => $varinfo)

{

if($varinfo->name == $name)

{

if(count($output) == 0 || ($varinfo->level > $output->level && $varinfo->level <= $level))

$output = $this->vartable[$index];

}

}

return $output;

}

function getFunc($level, $name, $params = -1)

{

$output = array();

foreach($this->functable as $index => $varinfo)

{

if($varinfo->name == $name && ($params == -1 || $varinfo->parameters == $params))

{

if(count($output) == 0 || ($varinfo->level > $output->level && $varinfo->level <= $level))

$output = $this->functable[$index];

}

}

return $output;

}

function leaveIfStatement($tree)

{

if($tree->tree['Expression']->type->index != 0)

{

$error = new CompileError($tree->token->file, $tree->token->line, $tree->token->col, 'Expression must be type Boolean (' . $tree->token->type . '): ' . $tree->token->content);

$error->error\_query();

}

}

function visitIdentifier($tree)

{

$var = $this->getVar($this->levelcounter, $tree->tree['Identifier']);

if(!is\_object($var))

{

$var = $this->getFunc($this->levelcounter, $tree->tree['Identifier']);

if(!is\_object($var))

{

$error = new CompileError($tree->token->file, $tree->token->line, $tree->token->col, 'Var does not exist (' . $tree->token->type . '): ' . $tree->token->content);

$error->error\_query();

}

else

{

$tree->info = $var;

}

}

else

{

$tree->info = $var;

}

}

function visitFunctionCall($tree)

{

$func = $this->getFunc($this->levelcounter, $tree->tree['FunctionName']);

if(!is\_object($func))

{

$error = new CompileError($tree->token->file, $tree->token->line, $tree->token->col, 'Undefined function (' . $tree->token->type . '): ' . $tree->token->content);

$error->error\_query();

}

else

{

$tree->info = $func;

}

}

function leaveForLoop($tree)

{

$varinfo = &$this->getVar($this->levelcounter, $tree->tree['Start']->tree['Assignment']['VariableName']->tree['Identifier']);

$varinfo->type = $tree->tree['Start']->tree['Assignment']['Expression']->type;

}

function leaveKeyword($tree)

{

// assign initial type

if($tree->tree['Keyword'] == 'v' || $tree->tree['Keyword'] == 'l')

{

$varinfo = &$this->getVar($this->levelcounter, $tree->tree['Declaration']->tree['Assignment']['VariableName']->tree['Identifier']);

$varinfo->type = $tree->tree['Declaration']->tree['Assignment']['Expression']->type;

}

}

function leaveExpression($tree)

{

//print\_r($tree->tree);

if(!isset($tree->tree['Operator']))

{

if(isset($tree->tree['Literal']->tree['BinaryLiteral'])) $tree->type = &$this->typetable[0];

if(isset($tree->tree['Literal']->tree['IntLiteral'])) $tree->type = &$this->typetable[1];

if(isset($tree->tree['Literal']->tree['RealLiteral'])) $tree->type = &$this->typetable[2];

if(isset($tree->tree['Literal']->tree['HexLiteral'])) $tree->type = &$this->typetable[3];

if(isset($tree->tree['Literal']->tree['StringLiteral'])) $tree->type = &$this->typetable[4];

}

elseif(!isset($tree->tree['VariableName']))

{

$type1 = 0;

if(isset($tree->tree['Literal']->tree['BinaryLiteral'])) $type1 = 0;

if(isset($tree->tree['Literal']->tree['IntLiteral'])) $type1 = 1;

if(isset($tree->tree['Literal']->tree['RealLiteral'])) $type1 = 2;

if(isset($tree->tree['Literal']->tree['HexLiteral'])) $type1 = 3;

if(isset($tree->tree['Literal']->tree['StringLiteral'])) $type1 = 4;

$type2 = 0;

if(isset($tree->tree['Expression']->tree['Literal']->tree['BinaryLiteral'])) $type2 = 0;

if(isset($tree->tree['Expression']->tree['Literal']->tree['IntLiteral'])) $type2 = 1;

if(isset($tree->tree['Expression']->tree['Literal']->tree['RealLiteral'])) $type2 = 2;

if(isset($tree->tree['Expression']->tree['Literal']->tree['HexLiteral'])) $type2 = 3;

if(isset($tree->tree['Expression']->tree['Literal']->tree['StringLiteral'])) $type2 = 4;

if(preg\_match('/^(\<|\>|==|\<=|\>=|\|\||&&|\!=)$/', $tree->tree['Operator']) != 0) $tree->type = &$this->typetable[0];

elseif($type1 > $type2) $tree->type = &$this->typetable[$type1];

else $tree->type = &$this->typetable[$type2];

}

else

{

$type1 = $tree->tree['VariableName']->type->index;

$type2 = 0;

if(isset($tree->tree['Expression']->tree['Literal']->tree['BinaryLiteral'])) $type2 = 0;

if(isset($tree->tree['Expression']->tree['Literal']->tree['IntLiteral'])) $type2 = 1;

if(isset($tree->tree['Expression']->tree['Literal']->tree['RealLiteral'])) $type2 = 2;

if(isset($tree->tree['Expression']->tree['Literal']->tree['HexLiteral'])) $type2 = 3;

if(isset($tree->tree['Expression']->tree['Literal']->tree['StringLiteral'])) $type2 = 4;

if(preg\_match('/^(\<|\>|==|\<=|\>=|\|\||&&|\!=)$/', $tree->tree['Operator']) != 0) $tree->type = &$this->typetable[0];

elseif($type1 > $type2) $tree->type = &$this->typetable[$type1];

else $tree->type = &$this->typetable[$type2];

}

}

}

class Collector extends Analyzer

{

function analyze()

{

$this->typetable = array(

new TypeInfo(0, 'BinaryLiteral', 1),

new TypeInfo(1, 'IntLiteral', 8),

new TypeInfo(2, 'RealLiteral', 16),

new TypeInfo(3, 'HexLiteral', 8),

new TypeInfo(4, 'StringLiteral', 8)

);

$this->functable[] = new FunctionInfo(0, 0, 'p', 1, array());

$this->visit($this->tree);

}

function visitParamDef($tree)

{

foreach($tree->tree['Params'] as $i => $identifier)

{

$this->vartable[] = new VarInfo(count($this->vartable), $tree->token, $this->levelcounter, $identifier->tree['Identifier']);

$newvar = &$this->vartable[count($this->vartable)-1];

$this->reverse\_vartable[$newvar->name . $newvar->level] = $newvar;

}

}

function visitFunctionDef($tree)

{

$this->functable[] = new FunctionInfo(count($this->functable), $this->levelcounter, $tree->tree['F-Name']->tree['Identifier'], count($tree->tree['ParamDef']->tree['Params']), $tree->tree['CommandList']);

}

function visitForLoop($tree)

{

// automatically transform this assignment into a declaration

$this->vartable[] = new VarInfo(count($this->vartable), $tree->token, $this->levelcounter, $tree->tree['Start']->tree['Assignment']['VariableName']->tree['Identifier']);

$newvar = &$this->vartable[count($this->vartable)-1];

$this->reverse\_vartable[$newvar->name . $newvar->level] = $newvar;

}

function visitKeyword($tree)

{

if($tree->tree['Keyword'] == 'v' || $tree->tree['Keyword'] == 'l')

{

$this->vartable[] = new VarInfo(count($this->vartable), $tree->token, $this->levelcounter, $tree->tree['Declaration']->tree['Assignment']['VariableName']->tree['Identifier']);

$newvar = &$this->vartable[count($this->vartable)-1];

$this->reverse\_vartable[$newvar->name . $newvar->level] = $newvar;

}

}

function leaveIfStatement($tree)

{

}

function visitFunctionCall($tree)

{

}

function leaveForLoop()

{

}

function leaveKeyword()

{

}

function leaveExpression()

{

}

function visitIdentifier()

{

}

}

class TypeInfo

{

var $name = '';

var $index = 0;

var $size = 8;

function TypeInfo($index, $name, $size)

{

$this->index = $index;

$this->name = $name;

$this->size = $size;

}

}

class VarInfo

{

var $level = 0;

var $token = array();

var $index = 0;

var $name = '';

function VarInfo($index, $token, $level, $name)

{

$this->level = $level;

$this->token = $token;

$this->index = $index;

$this->name = $name;

}

}

class FunctionInfo

{

var $level = 0;

var $name = '';

var $index = 0; // signature

var $parameters = 0;

var $body = array();

function FunctionInfo($index, $level, $name, $parameters, $body)

{

$this->index = $index;

$this->level = $level;

$this->name = $name;

$this->parameters = $parameters;

$this->body = $body;

}

}

?>