McGill University
School of Computer Science
COMP 520

Wig Compiler Report

Report No. 2003-05

Gregory Pekofsky
Eric Fong
Maxime Chambreuil

November 24, 2003

Contents

1 Introduction

Most of wig compilers have produced and still produce C code. This perpective of doing what everybody has done, didn't motivate us and we have found that it would be fun to generate PHP code, as we are familiar with this language.

1.1 Clarifications

After studying the WIG language during few weeks, we didn't discover any unclear points and even during the implementation we didn't meet any problems with the language definition.

1.2 Restrictions

There is no restriction in our version of WIG. PHP enabled us to forget some problem, like empty variable, which is considered as an empty string by PHP. This is just an example to show you that because of our output, we have a more flexible compiler.

1.3 Extensions

When no service is specified, the output of our compiler proposes a list of available services.

1.4 Implementation Status

The implementation of our compiler is over. We have tested it with every possible benchmarks and it works really good.

2 Parsing and Abstract Syntax Trees

2.1 Using the `SableCC` Tool

We have implemented the WIG scanner and parser using SableCC2 using the grammar on the course website, except we made the following change(s) which does not effect the wig language, but does effect implementation.

- We got rid of "inputattr" and "inputtype" as they aren't really needed because they are themselves html attributes.
- We distinguished attribute values by tagstr,tagint, and percent. This hopefully will be useful later in implementation.
- We added // and /**/ type comments, which are only affective outside an html variable.

2.2 Abstract Syntax Trees

Since we are using SableCC2 and not SableCC3, due to the bug that our group found and informed the professor about and since this is our first time coding a compiler we have decided not to use an AST so that we can code directly from the grammar.

2.3 Desugaring

How did we handle Meta Tags: We used a similar idea to finding /*...*/. As well we used states as Meta Tags are only allowed in html variables.

```
'<!--' ( [all-'-']* | '->' |  '-'+ [[all-'-']-'>']  )* '-'* '-->';/
```

2.4 Weeding

During the weeding walk in the tree, we only check if there is a return statement inside a schema, which is not allowed.

2.5 Testing

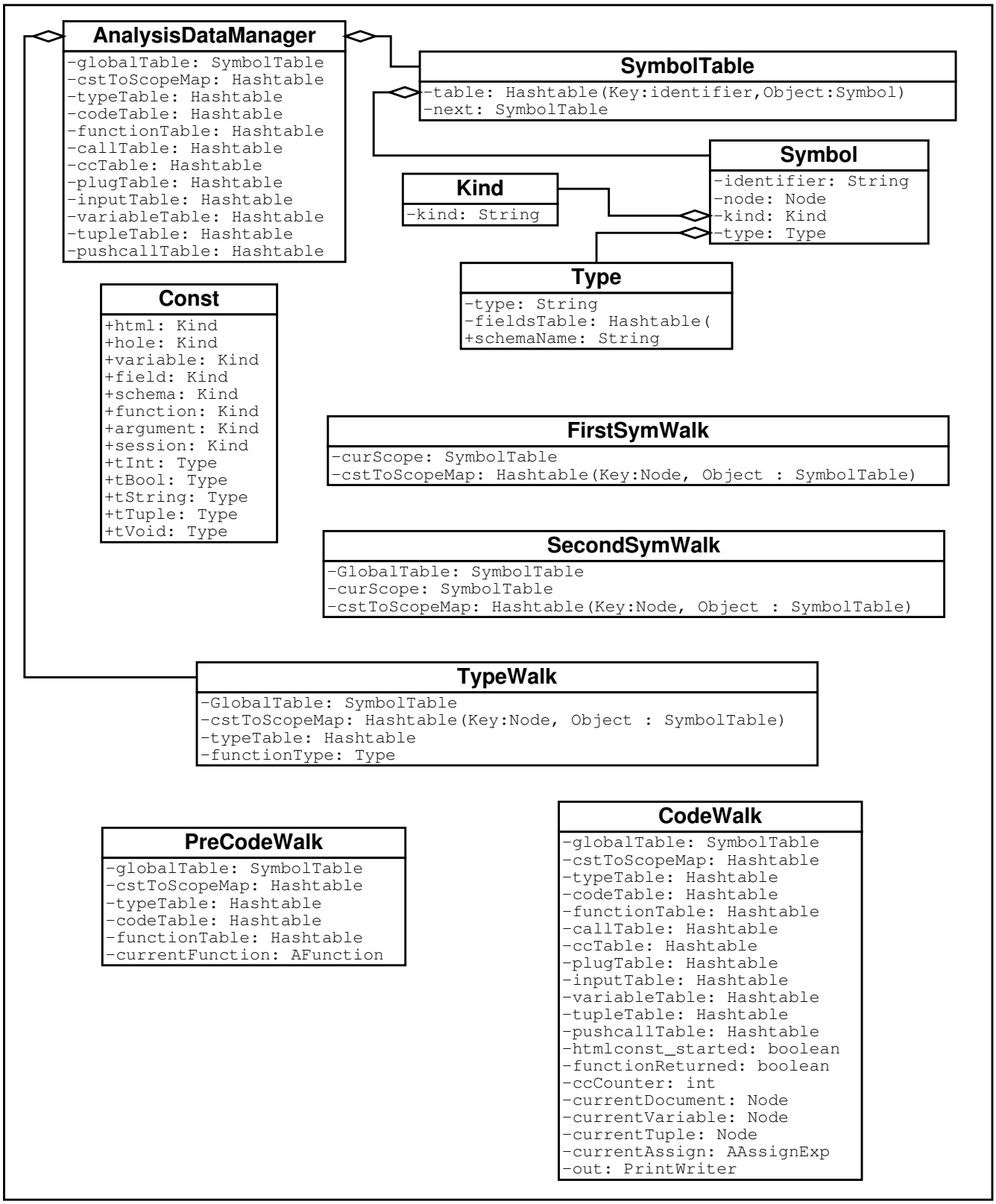
We have implemented a Pretty Printer, even though it is not very pretty. Though, we have tested all the 2002 benchmarks, which we had to expand our language to suit them because they didn't follow the wig grammar perfectly. So our compiler will incorporate all these changes.

How did we test the benchmarks:

- pretty(parse(pretty(parse(X))))= pretty(parse(X)). There were no syntatical errors.
- As well, we ran each output through wig10

3 Symbol Tables

This is the UML classes diagram we have used for symbol tables, type-checking and code generation:


AnalysisDataManager
-globalTable: SymbolTable
-cstToScopeMap: Hashtable
-typeTable: Hashtable
-codeTable: Hashtable
-functionTable: Hashtable
-callTable: Hashtable
-ccTable: Hashtable
-plugTable: Hashtable
-inputTable: Hashtable
-variableTable: Hashtable
-tupleTable: Hashtable
-pushcallTable: Hashtable
SymbolTable
-table: Hashtable(Key:identifier,Object:Symbol)
-next: SymbolTable
Symbol
-identifier: String
-node: Node
-kind: Kind
-type: Type
Kind
-kind: String
Type
-type: String
-fieldsTable: Hashtable(
+schemaName: String
Const
+html: Kind
+hole: Kind
+variable: Kind
+field: Kind
+schema: Kind
+function: Kind
+argument: Kind
+session: Kind
+tInt: Type
+tBool: Type
+tString: Type
+tTuple: Type
+tVoid: Type
FirstSymWalk
-curScope: SymbolTable
-cstToScopeMap: Hashtable(Key:Node, Object : SymbolTable)
SecondSymWalk
-GlobalTable: SymbolTable
-curScope: SymbolTable
-cstToScopeMap: Hashtable(Key:Node, Object : SymbolTable)
TypeWalk
-GlobalTable: SymbolTable
-cstToScopeMap: Hashtable(Key:Node, Object : SymbolTable)
-typeTable: Hashtable
-functionType: Type
PreCodeWalk
-globalTable: SymbolTable
-cstToScopeMap: Hashtable
-typeTable: Hashtable
-codeTable: Hashtable
-functionTable: Hashtable
-currentFunction: AFunction
CodeWalk
-globalTable: SymbolTable
-cstToScopeMap: Hashtable
-typeTable: Hashtable
-codeTable: Hashtable
-functionTable: Hashtable
-callTable: Hashtable
-ccTable: Hashtable
-plugTable: Hashtable
-inputTable: Hashtable
-variableTable: Hashtable
-tupleTable: Hashtable
-pushcallTable: Hashtable
-htmlconst_started: boolean
-functionReturned: boolean
-ccCounter: int
-currentDocument: Node
-currentVariable: Node
-currentTuple: Node
-currentAssign: AAssignExp
-out: PrintWriter

At this point, we have defined the kinds of identifiers that occur in WIG programs:

- html
- hole
- variable
- field
- schema
- function
- argument
- session

and we have decided what symbol information is relevant to each kind:

- html: name
- hole: name
- variable: name and type
- field: name and type
- schema: name
- function: name
- argument: name and type
- session: name

We have implemented the construction of symbol tables with 2 passes: The first walk collects general information: inAHtml, inASession, inAFunction, inASchema, etc... The second deals with information deeper in the tree: inAPlug, inAManyIdentifiers, inAOneIdentifiers, inAArgument, inACompoundstm, inAReceive, etc...

3.1 Symbol Data

As you can notice from the diagram, a Symbol Table entry consists in a Symbol containing the identifier, the node in the tree, the kind of symbol (html, hole, variable, field, etc...) and the type (tInt, tBool, etc...). We can access the Symbol in the Symbol Table with the identifier.

3.2 Algorithm

During the first walk of symbol table, we build the global table and the hashtable cstToScopeMap. The global table contains the symbol of html constants, function, session, schema. The hashtable cstToScopeMap contains the node and the symbol table associated to this node.

During the second walk, we have finished building the cstToScopeMap hashtable with the symbol that can be found inside a scope: variable, argument, tuple, plug, receive,etc... We have also added those symbol to the symbol table associated to the scope.

3.3 Testing

To test this phase, we have run the compiler on benchmarks and displayed the Symbol Table at the end to compare it with the source of the benchmark. Every Symbol Table matches correctly the associated scope in the code.

4 Type Checking

4.1 Types

Our language supports integer, boolean, string, tuple and void type.

4.2 Type Rules

Concerning the type check walk, we performs differents type checking. For example :

- We check if the type of the returned variable matches the type of the function : The functionType is initialized when we get in the function and the type check is done when we get out of the function.
- We check if the type and the number of variables in a function call matches the signature of the function.

4.3 Algorithm

To check type we have implemented a new walk to perform the type check we have described previously.

4.4 Testing

We have tested our implementation by annotating statements with types in our pretty printer. We have run benchmarks containing errors we wanted to catch: we received type errors for each of them.

5 Code Generation

5.1 Strategy

To generate code, we have use 2 walks. The first walk performs the construction of a linked list of formal parameters of a function. The second does the following:

- Build up Code Table
- callTable is a hash table with keys=nodes and values=linked list where the links list is the orders of arguments in a call. Each value in the linked list is a string of the expression
- ccTable is used in the if-else while statements to keep track of where in the compound statement there are
- plugTable is similar to ccTable and functionTable ... this is meant for show and exit
- inputTable is similar to ccTable and functionTable ... this is meant to show receive
- variableTable is similar to ccTable and functionTable ... but is meant for variable*
- tupleTable

- pushcallTable ... linked the function push to its statement, so that its printed just before the statement. It contains a linked list of strings
- in codeTable when I have a string that begins with a "@" it means that its tuple assignmenet. I get rid of the "@" when I print to file

5.2 Code Templates

This is the PHP code, which is included at the beginning of each generated file:

```
<?
/*
 * Start or resume the session
 */
session_start();
session_register("stack");
session_register("retstack");
session_register("started");
session_register("php_self_old");
session_register("session_old");
$stack = $_SESSION["stack"];
$retstack = $_SESSION["retstack"];
$started = $_SESSION["started"];
$php_self_old = $_SESSION["php_self_old"];
$session_old = $_SESSION["session_old"];

/*
 * Assign 0 if the input is null string
 */
foreach($_POST as $key=>$value)
{
    if($_POST[$key]=="")
    {
        $_POST[$key] = 0;
    }
}

/*
 * Init the Environment Variables and Array
 */
$PHP_SELF = explode("/",$_SERVER["PHP_SELF"]);
$PHP_SELF = $PHP_SELF[count($PHP_SELF)-1];
$filepath=$PHP_SELF.".globals";
$firstsession = $_SERVER["QUERY_STRING"];
$exit = false;

/*
 * Load Global Variables
 */
if(is_file($filepath))
{
    $file=fopen($filepath, 'r');
}
```

```
// Read content of the array from the opened file.
if(is_readable($filepath))
{
    $filestat=fstat($file);
    $contents = fread($file,$filestat[7]);

    $globals=unserialize($contents);
    fclose($file);
}


// Reset the all the variables if new session are loaded
if(!is_array($globals))
{
    $globals = array();  //file
}
if(!is_array($stack) || $php_self_old != $PHP_SELF || $firstsession != $session_old)
{
    $stack = array();  //session
    $retstack = array(); //session
}

// Arry for store Html Const
$htmlconst = array();


/*
 * Return result from function to the session or function
 */
function setResult($res)
{
    global $retstack;
    global $stack;
    array_pop($stack);
    array_push($retstack,$res);
}

/*
 * Get the result from the function
 */
function getResult()
{
    global $retstack;
    global $stack;
    return array_pop($retstack);
}

/*
 * Show the Html Const with plug-in variables
 */
function plugHtml($vararray, $html)
{
    global $PHP_SELF;
```

```
global $firstsession;
global $exit;
foreach($vararray as $key => $value)
{
    $key = "<[".$key."]>";
    if(gettype($value)=="boolean")
    {
        if($value == true)
            $value = "true";
        else
            $value = "false";
    }
    $html = str_replace($key,$value,$html);
}
?><html><?
if(!$exit)
{
    ?><form method=post action="<?=$PHP_SELF?>?<?=$firstsession?>"><?
                            }
                            print($html);
if(!$exit)
{
    ?><br><br><input type=submit value=Continue></form><?
                                    }
                                    ?></html><?

                                    global $_SESSION;
global $globals;
global $filepath;
global $stack;
global $retstack;
if($exit)
{
    array_pop($stack);
}
$_SESSION["stack"] = $stack;
$_SESSION["retstack"] = $retstack;
$_SESSION["started"] = 1;
$_SESSION["php_self_old"] = $PHP_SELF;
$_SESSION["session_old"] = $firstsession;

/* save the global variable to file */
if(!$file = fopen($filepath, "w"))
{
    print("Can't open the globals data file");
    return;
}

// Write content of the array to the opened file.
if(is_writable($filepath))
{
    fwrite($file,serialize($globals));
}
```

```
    else
    {
        print("Can't write to the global data file");
        return;
    }
    fclose($file);

    exit();
}
?>
```

5.3 Algorithm

As we said before, we have used 2 walks of the tree to generate the code.

5.4 Runtime System

The runtime system used is the Apache web server and the PHP module.

5.5 Sample Code

This is the complete code generated for the service `tiny.wig`:

```
<?
/*
 * Start or resume the session
 */
session_start();
session_register("stack");
session_register("retstack");
session_register("started");
session_register("php_self_old");
session_register("session_old");
$stack = $_SESSION["stack"];
$retstack = $_SESSION["retstack"];
$started = $_SESSION["started"];
$php_self_old = $_SESSION["php_self_old"];
$session_old = $_SESSION["session_old"];

/*
 * Assign 0 if the input is null string
 */
foreach($_POST as $key=>$value)
{
    if($_POST[$key]=="")
    {
        $_POST[$key] = 0;
    }
}

/*
 * Init the Environment Variables and Array
```

```
 */
$PHP_SELF = explode("/",$_SERVER["PHP_SELF"]);
$PHP_SELF = $PHP_SELF[count($PHP_SELF)-1];
$filepath=$PHP_SELF.".globals";
$firstsession = $_SERVER["QUERY_STRING"];
$exit = false;

/*
 * Load Global Variables
 */
if(is_file($filepath))
{
    $file=fopen($filepath, 'r');
}

// Read content of the array from the opened file.
if(is_readable($filepath))
{
    $filestat=fstat($file);
    $contents = fread($file,$filestat[7]);

    $globals=unserialize($contents);
    fclose($file);
}


// Reset the all the variables if new session are loaded
if(!is_array($globals))
{
    $globals = array();  //file
}
if(!is_array($stack) || $php_self_old != $PHP_SELF || $firstsession != $session_old)
{
    $stack = array();  //session
    $retstack = array(); //session
}

// Arry for store Html Const
$htmlconst = array();


/*
 * Return result from function to the session or function
 */
function setResult($res)
{
    global $retstack;
    global $stack;
    array_pop($stack);
    array_push($retstack,$res);
}

/*
```

```
 * Get the result from the function
 */
function getResult()
{
    global $retstack;
    global $stack;
    return array_pop($retstack);
}

/*
 * Show the Html Const with plug-in variables
 */
function plugHtml($vararray, $html)
{
    global $PHP_SELF;
    global $firstsession;
    global $exit;
    foreach($vararray as $key => $value)
    {
        $key = "<[".$key."]>";
        if(gettype($value)=="boolean")
        {
            if($value == true)
                $value = "true";
            else
                $value = "false";
        }
        $html = str_replace($key,$value,$html);
    }
    ?><html><?
    if(!$exit)
    {
        ?><form method=post action="<?=$PHP_SELF?>?<?=$firstsession?>"><?
                                }
                                print($html);
    if(!$exit)
    {
        ?><br><br><input type=submit value=Continue></form><?
                                        }
                                        ?></html><?

                                        global $_SESSION;
    global $globals;
    global $filepath;
    global $stack;
    global $retstack;
    if($exit)
    {
        array_pop($stack);
    }
    $_SESSION["stack"] = $stack;
    $_SESSION["retstack"] = $retstack;
    $_SESSION["started"] = 1;
```

```
    $_SESSION["php_self_old"] = $PHP_SELF;
    $_SESSION["session_old"] = $firstsession;

    /* save the global variable to file */
    if(!$file = fopen($filepath, "w"))
    {
        print("Can't open the globals data file");
        return;
    }

    // Write content of the array to the opened file.
    if(is_writable($filepath))
    {
        fwrite($file,serialize($globals));
    }
    else
    {
        print("Can't write to the global data file");
        return;
    }
    fclose($file);

    exit();
}
?>


<?

/*Declare Global Variables*/
if($started!=1)
{
    $globals["amount"];
}

ob_start();
?>
<body >
Welcome!
</body>
<?
$htmlconst["Welcome"] = ob_get_contents();
ob_clean();
?>
<body >
How much do
    you want to contribute?
    <input name="contribution" type="text" size =4 >
                                    </body>
                                    <?
                                    $htmlconst["Pledge"] = ob_get_contents();
ob_clean();
?>
```

```
<body >
The total is now  <[total]>.
</body>
<?
$htmlconst["Total"] = ob_get_contents();
ob_clean();

class session_Contribute
{
    //$cc
    //$local
    //$name
    function session_Contribute()
    {
        if(!isset($this->cc))
            $this->cc = array();
        if(!isset($this->local))
            $this->local = array();
    }
    function run()
    {
        global $globals;
        global $_POST;
        global $htmlconst;
        global $exit;
        global $stack;
        if(!$this->cc[0])
        {
            $this->local["i"];
            $this->cc[0] = true;
        }
        if(!$this->cc[1])
        {
            $this->local["i"]=87;
            $this->cc[1]=true;
        }
        if(!$this->cc[2])
        {
            $this->cc[2] = true;
            $plugarray = array();
            plugHtml($plugarray,$htmlconst["Welcome"]);
        }
        if(!$this->cc[3])
        {
            $this->cc[3] = true;
        }
        if(!$this->cc[4])
        {
            $this->cc[4] = true;
            $plugarray = array();
            plugHtml($plugarray,$htmlconst["Pledge"]);
        }
        if(!$this->cc[5])
```

```
        {
            $this->cc[5] = true;
            $this->local["i"] = $_POST["contribution"];
        }
        if(!$this->cc[6])
        {
            $globals["amount"]=$globals["amount"]+$this->local["i"];
            $this->cc[6]=true;
        }
        if(!$this->cc[7])
        {
            $exit = true;
            $this->cc[7] = true;
            $plugarray = array();
            $plugarray["total"] = $globals["amount"];
            plugHtml($plugarray,$htmlconst["Total"]);
        }

        array_pop($stack);

    }
}
?>


<?
/*code to manage starting the function on top of the stack*/
if(count($stack)==0)
{
    if(class_exists("session_".$firstsession))
    {
        $code = "array_push(\$stack, new session_$firstsession());";
        eval($code);
        $stack[count($stack)-1]->run();
        while(count($stack)>0)
        {
            $stack[count($stack)-1]->run();
        }
    }
    else
    {
        ?><html><body>
        NO SESSION SELECTED!
        <p>Please Select the following Session:<br>
        <a href="<?=$PHP_SELF?>?Contribute"><?=$PHP_SELF?>?Contribute<br>
                                            </body></html><?
                                        }
                                    }
                                    else
                                    {
                                        while(count($stack)>0)
                                        {
                                            $stack[count($stack)-1]->run();
```

```
        }
    }
    ?>
```

5.6 Testing

We have tested this phase by compiling every available benchmarks and calling every services for each of them.

6 Availability and Group Dynamics

6.1 Manual

You can run "make" to compile and "run" to test the test file. The php files are in output directory (not ./output/testfiles): Some tools are used to do the indentation to make them easy to read.

6.2 Demo Site

Give the URL for a web site that contains demos of services that you have generated.

http://www.cs.mcgill.ca/ ffong/

6.3 Division of Group Duties

6.3.1 Gregory Pekofsky

I did:

- a part of change to the grammar
- the weed walk
- a part of the symbol table walk
- a part of the type walk
- a part of the code generation walk
- the tests

6.3.2 Eric Fong

I did:

- a part of change to the grammar
- a part of the symbol table walk
- a part of the type walk
- a part of the code generation walk
- the tests
- specialize on the tuple

6.3.3 Maxime Chambreuil

I did:

- the first milestone
- a part of change to the grammar
- a part of the symbol table walk
- a part of the type walk
- the tests
- this report
- the administration of the CVS server
- the support for using CVS