GROUP-H

Wig Compiler

COMP 520 Compiler Design Final Report

Dwijesh Bhageerutty
Faiz Khan
Amine Sahibi

12/4/2012

1. INTRODUCTION

1.1 CLARIFICATIONS

The target language for code generation is PHP. Please take a look at the list of keywords for PHP and be careful not using them as variable names. [See section 8.5]

1.2 RESTRICTIONS

- We do not support show/exit statements in functions.
- HTML tags in this format are not accepted: “<[^>]*/>”, e.g.: “<input name = “name” type=”text” **/**>”
- Also see the section 2.6, cases WC11 and WC12.

1.3 EXTENSIONS

No extensions

1.4 IMPLEMENTATION STATUS

All the basic features for the WIG compiler have been implemented. This includes tuple operations - creation, keep, join, remove, keep many, remove many, comparison and assignment. We also have chained tuple operations, except for some cases (See restrictions).

We generate PHP scripts for the code generation part of the project.

2 PARSING AND ABSTRACT SYNTAX TREES

2.1 THE GRAMMAR

```
Package wig;

/***********************************
* Helpers
***********************************/
Helpers

        any = [0..0xffff];

        letter = [['a'..'z'] + ['A'..'Z']];
        digit = ['0'..'9'];
        letter_or_digit = letter | digit;
        letter_or_digit_or_us = letter_or_digit | '_';
        letter_or_us = letter | '_';

        tab = 9;
        cr = 13;
        lf = 10;
        eol = lf|cr|cr lf;

        no_dash = [any - '-'];
        no_meta_stm = [no_dash -'>'];
        no_quote = [any-'"'];
        no_star = [any-'*'];
        no_star_backslash = [no_star -'/'];
        no_cr_lf = [any - [lf+cr]];

        escapequote = '\"';

        meta_start = '<!--';
        meta_end = '-->';
        meta_stmt = meta_start no_dash* '-'+ (no_meta_stm no_dash* '-'+)* meta_end;

        string_const = '"' (no_quote* escapequote* no_quote*)* '"';

        inline_comment = '//' no_cr_lf* eol;
        block_comment = '/*' no_star* '*'+(no_star_backslash no_star* '*' +)* '/';
        wig_comment = inline_comment|block_comment;

        whatever_stm = [[any-'<']-'>']*; //anything without < and >
```

```
/***********************************
* States
************************************/
States

        wig_scope,html_scope,whatever_scope,html_tag_scope, hole_scope;

/***********************************
* Tokens
************************************/
Tokens

/*********************
* Wig Keywords
*********************/
        {wig_scope} service = 'service';
        {wig_scope} const = 'const';
        {wig_scope} html = 'html';
        {wig_scope->html_scope,html_scope} html_tag_start = '<html>';
        {wig_scope} schema = 'schema';
        {wig_scope} session = 'session';
        {wig_scope} show = 'show';
        {wig_scope} exit = 'exit';
        {wig_scope} return = 'return';
        {wig_scope} if = 'if';
        {wig_scope} else = 'else';
        {wig_scope} while = 'while';
        {wig_scope} plug = 'plug';
        {wig_scope} receive = 'receive';
        {wig_scope} int = 'int';
        {wig_scope} bool = 'bool';
        {wig_scope} string = 'string';
        {wig_scope} void = 'void';
        {wig_scope} tuple = 'tuple';
        {wig_scope} true = 'true';
        {wig_scope} false = 'false';


/*********************
* HTML Keywords
*********************/
        {html_scope} meta = meta_stmt; // any string of the form <!-- ... -->
        {html_scope->wig_scope,whatever_scope->html_scope,wig_scope} html_tag_end = '</html>';
        {html_scope->html_tag_scope} input = 'input';
        {html_scope,html_tag_scope,wig_scope} pos_intconst = '0' | [digit-'0'] digit*;
        {html_scope} neg_intconst = '-' [digit-'0'] digit*;
        {html_scope} select = 'select';
```

```
/*********************
* HTML Tags (Input) Keywords
*********************/
        {html_tag_scope} type = 'type';
        {html_tag_scope} name = 'name';
        {html_tag_scope} text = 'text';
        {html_tag_scope} radio = 'radio';

/*********************
* Symbols & Operators
*********************/
        {wig_scope} l_brace = '{';
        {wig_scope} r_brace = '}';
        {wig_scope,html_scope,html_tag_scope} assign = '=';
        {wig_scope, html_scope->wig_scope} semicolon = ';';
        {wig_scope,whatever_scope->html_scope,html_scope} lt = '<';
        {wig_scope,html_scope->whatever_scope,html_tag_scope->whatever_scope} gt = '>';
        {whatever_scope->html_scope,html_scope} lt_slash = '</';
        {whatever_scope->hole_scope, html_scope} lt_bracket = '<[';
        {hole_scope->whatever_scope, html_scope} gt_bracket = ']>';
        {wig_scope} comment = wig_comment;
        {wig_scope} l_par = '(';
        {wig_scope} r_par = ')';
        {wig_scope} l_bracket = '[';
        {wig_scope} r_bracket = ']';
        {wig_scope} comma = ',';
        {wig_scope} keep = '\+';
        {wig_scope} remove = '\-';
        {wig_scope} join = '<<';
        {wig_scope} eq = '==';
        {wig_scope} neq = '!=';
        {wig_scope} lteq = '<=';
        {wig_scope} gteq = '>=';
        {wig_scope} not = '!';
        {wig_scope} minus = '-';
        {wig_scope} plus = '+';
        {wig_scope} mult = '*';
        {wig_scope} div = '/';
        {wig_scope} mod = '%';
        {wig_scope} and = '&&';
        {wig_scope} or = '||';
        {wig_scope} dot = '.';
```

```
/*********************
* Others
*********************/
        {wig_scope,html_scope,html_tag_scope} eol = eol;
        {wig_scope,html_scope,html_tag_scope} blank = (tab|' '|eol)*;
        {wig_scope,html_scope,html_tag_scope, hole_scope} identifier = letter_or_us (letter_or_us|digit)*; // usual
identifiers
        {wig_scope, html_tag_scope, html_scope} stringconst = string_const ; // usual string constants

        {whatever_scope} whatever = whatever_stm; // any string not containing < or >

/*********************
* Ignored Tokens *
*********************/
Ignored Tokens
        blank, comment, eol;


/***********************************
* Productions
***********************************/
Productions

  service = T.service l_brace P.html+ P.schema* variable* function* P.session+ r_brace
    {-> New service([html], [schema], [variable], [function], [session])};

  html = const T.html identifier assign html_tag_start htmlbody* html_tag_end semicolon
    {-> New html(identifier, [htmlbody])};

  htmlbody{->htmlbody} =
   {tag_start} lt identifier attribute* gt
    {->New htmlbody.tag_start(identifier, [attribute])}
  | {tag_end} lt_slash identifier gt
    {->New htmlbody.tag_end(identifier)}
  | {hole} lt_bracket identifier gt_bracket
    {->New htmlbody.hole(identifier)}
  | {whatever} whatever
    {->New htmlbody.whatever(whatever)}
  | {meta} meta
    {->New htmlbody.meta(meta)}
  | {input} lt T.input inputattr+ gt
    {->New htmlbody.input(T.input, [inputattr])}
  | {select} lt [select_tag]:select inputattr+ [first_gt]:gt htmlbody* lt_slash select [second_gt]:gt
    {->New htmlbody.select(select_tag, [inputattr], first_gt, [htmlbody])};

  inputattr{->inputattr} =
   {name} name assign attr
    {->New inputattr.name(name, attr.attr)}
  | {type} T.type assign inputtype
```

```
   {->New inputattr.type(T.type, inputtype)}
| {attribute} attribute
   {->New inputattr.attribute(attribute)};

inputtype{->inputtype} =
 {texttype} text
   {->New inputtype.texttype(text)}
| {radiotype} radio
   {->New inputtype.radiotype(radio)}
| {strtype} stringconst
   {->New inputtype.strtype(stringconst)};

attribute {->attribute} =
 {attr} attr
   {->New attribute.attr(attr.attr)}
| {assign} [left_attr]:attr assign [right_attr]:attr
   {->New attribute.assign(left_attr.attr, right_attr.attr)};

attr{->attr} =
 {id} identifier
   {->New attr.id(identifier)}
| {str} stringconst
   {->New attr.str(stringconst)}
| {iconst} P.intconst
   {->New attr.iconst(intconst)};

intconst{->P.intconst} =
 {negint} neg_intconst
   {->New intconst.negint(neg_intconst)}
| {posint} pos_intconst
   {->New intconst.posint(pos_intconst)};

schema = T.schema identifier l_brace field* r_brace
   {->New schema( identifier, [field])};

field{->field} = simpletype identifier semicolon
   {-> New field(simpletype.type, identifier)};

variable{->variable} = P.type identifiers semicolon
   {->New variable(P.type, [identifiers.identifier])};

identifiers{->identifier*} =
 {one} identifier
   {->[identifier]}
| {many} identifiers comma identifier
   {-> [identifiers.identifier, identifier]};

simpletype{->P.type} =
 {int} int
   {->New type.int(int)}
```

```
| {bool} bool
   {->New type.bool(bool)}
| {string} string
   {->New type.string(string)}
| {void} void
   {->New type.void(void)};

type{->P.type} =
  {simple} simpletype
   {->simpletype.type}
| {tuple} tuple identifier
   {->New type.tuple(identifier)};

function{->function} = P.type identifier l_par arguments? r_par compoundstm
   {->New function(P.type, identifier, [arguments.argument], compoundstm)};

arguments{->argument*} =
  {one} argument
   {->[argument]}
| {many} arguments comma argument
   {-> [arguments.argument, argument]};

argument{->argument} = P.type identifier
   {->New argument(P.type, identifier)};

session = T.session identifier l_par r_par compoundstm
   {->New session(identifier, compoundstm)};

stm{->stm?} =
  {empty} semicolon
   {->New stm.empty()}
| {show} show document P.receive? semicolon
   {-> New stm.show(document, P.receive)}
| {exit} exit document semicolon
   {-> New stm.exit(document)}
| {return} return semicolon
   {-> New stm.return()}
| {returnexp} return exp semicolon
   {-> New stm.returnexp(exp)}
| {if} if l_par exp r_par stm
   {-> New stm.if(exp, stm)}
| {ifelse} if l_par exp r_par [then_stm]:stm_no_short_if else [else_stm]:stm
   {-> New stm.ifelse(exp, then_stm.stm, else_stm)}
| {while} while l_par exp r_par stm
   {-> New stm.while(exp, stm)}
| {comp} compoundstm
   {-> New stm.comp(compoundstm)}
| {exp} exp semicolon
   {-> New stm.exp(exp)};
```

```
stm_no_short_if{->stm} =
  {empty} semicolon
    {->New stm.empty() }
| {show} show document P.receive? semicolon
    {->New stm.show(document, P.receive) }
| {exit} exit document semicolon
    {-> New stm.exit(document)}
| {return} return semicolon
    {-> New stm.return()}
| {returnexp} return exp semicolon
    {-> New stm.returnexp(exp)}
| {ifelse} if l_par exp r_par [then_stm]:stm_no_short_if else [else_stm]:stm_no_short_if
    {-> New stm.ifelse(exp, then_stm.stm, else_stm.stm)}
| {while} while l_par exp r_par stm_no_short_if
    {-> New stm.while(exp, stm_no_short_if.stm)}
| {comp} compoundstm
    {-> New stm.comp(compoundstm)}
| {exp} exp semicolon
    {-> New stm.exp(exp)};

document =
  {id} identifier
    {-> New document.id(identifier)}
| {plug} T.plug identifier l_bracket plugs r_bracket
    {-> New document.plug(identifier, [plugs.plug])};

receive = T.receive l_bracket inputs r_bracket
    {-> New receive([inputs.input])};

compoundstm = l_brace variable* stm* r_brace
    {-> New compoundstm([variable], [stm])};

plugs{->P.plug*} =
  {one} P.plug
    {->[P.plug]}
| {many} P.plugs comma P.plug
    {-> [plugs.plug, P.plug]};

plug = identifier assign exp
    {-> New plug(identifier, exp.exp)};

inputs{->P.input*} =
  {one} P.input
    {-> [P.input]}
| {many} P.inputs comma P.input
    {-> [inputs.input, input]};

input = lvalue assign identifier
    {-> New input(lvalue, identifier)};
```

```
exp{->exp} =
  {assign} lvalue assign [right]:or_exp
    {-> New exp.assign(lvalue, right.exp)}
| {default} [left]:or_exp
    {-> left.exp};

or_exp{->exp} =
  {or} [left]:or_exp or [right]:and_exp
    {-> New exp.or(left.exp, right.exp)}
| {default} [left]:and_exp
    {-> left.exp};

and_exp{->exp} =
  {and} [left]:and_exp and [right]:cmp_exp
    {-> New exp.and(left.exp, right.exp)}
| {default} [left]:cmp_exp
    {-> left.exp};

cmp_exp{->exp} =
  {eq} [left]:add_exp eq [right]:add_exp
    {-> New exp.eq(left.exp, right.exp)}
| {neq} [left]:add_exp neq [right]:add_exp
    {-> New exp.neq(left.exp, right.exp)}
| {lt} [left]:add_exp lt [right]:add_exp
    {-> New exp.lt(left.exp, right.exp)}
| {gt} [left]:add_exp gt [right]:add_exp
    {-> New exp.gt(left.exp, right.exp)}
| {lteq} [left]:add_exp lteq [right]:add_exp
    {-> New exp.lteq(left.exp, right.exp)}
| {gteq} [left]:add_exp gteq [right]:add_exp
    {-> New exp.gteq(left.exp, right.exp)}
| {default} [left]:add_exp {-> left.exp};

add_exp{->exp} =
  {plus} [left]:add_exp plus [right]:mult_exp
    {-> New exp.plus(left.exp, right.exp)}
| {minus} [left]:add_exp minus [right]:mult_exp
    {-> New exp.minus(left.exp, right.exp)}
| {default} [left]:mult_exp
    {-> left.exp};

mult_exp{->exp} =
  {mult} [left]:mult_exp mult [right]:join_exp
    {-> New exp.mult(left.exp, right.exp)}
| {div} [left]:mult_exp div [right]:join_exp
    {-> New exp.div(left.exp, right.exp)}
| {mod} [left]:mult_exp mod [right]:join_exp
    {-> New exp.mod(left.exp, right.exp)}
| {default} [left]:join_exp
    {-> left.exp};
```

```
join_exp{->exp} =
  {join} [left]:tuple_exp join [right]:join_exp
    {-> New exp.join(left.exp, right.exp)}
| {default} [left]:tuple_exp
    {-> left.exp};

tuple_exp{->exp} =
  {keep} [left]:tuple_exp keep identifier
    {-> New exp.keep(left.exp, identifier)}
| {remove} [left]:tuple_exp remove identifier
    {-> New exp.remove(left.exp, identifier)}
| {keep_many} [left]:tuple_exp keep l_par identifiers r_par
    {-> New exp.keep_many(left.exp, [identifiers.identifier])}
| {remove_many} [left]:tuple_exp remove l_par identifiers r_par
    {-> New exp.remove_many(left.exp, [identifiers.identifier])}
| {default} [left]:unary_exp
    {-> left.exp};

unary_exp{->exp} =
  {not} not [left]:base_exp
    {-> New exp.not(left.exp)}
| {neg} minus [left]:base_exp
    {-> New exp.neg(left.exp)}
| {default} [left]:base_exp
    {-> left.exp};

base_exp{->exp} =
  {lvalue} lvalue
    {-> New exp.lvalue(lvalue)}
| {call} identifier l_par exps? r_par
    {-> New exp.call(identifier, [exps.exp])}
| {int} P.intconst
    {-> New exp.int(intconst)}
| {true} true
    {-> New exp.true(true)}
| {false} false
    {-> New exp.false(false)}
| {string} stringconst
    {-> New exp.string(stringconst)}
| {tuple} tuple l_brace fieldvalues? r_brace
    {-> New exp.tuple([fieldvalues.fieldvalue])}
| {paren} l_par exp r_par
    {-> exp.exp};

exps{->exp*} =
  {one} exp
    {-> [exp]}
| {many} exps comma exp
    {-> [exps.exp, exp]};
```

```
  lvalue =
    {simple} identifier
      {-> New lvalue.simple(identifier)}
  | {qualified} [left]:identifier dot [right]:identifier
      {-> New lvalue.qualified(left, right)};

  fieldvalues{->fieldvalue*} =
    {one} fieldvalue
      {-> [fieldvalue]}
  | {many} fieldvalues comma fieldvalue
      {-> [fieldvalues.fieldvalue, fieldvalue]};

  fieldvalue = identifier assign exp
      {-> New fieldvalue(identifier, exp)};

/***********************************
* Abstract Syntax Tree
***********************************/
Abstract Syntax Tree

  service = P.html+ P.schema* variable* function* P.session+;

  html = identifier htmlbody*;

  htmlbody =
       {tag_start} identifier attribute*
     | {tag_end} identifier
     | {hole} identifier
     | {whatever} whatever
     | {meta} meta
     | {input} T.input inputattr+
     | {select} [select_tag]:select inputattr+ [first_gt]:gt htmlbody*;

  inputattr =
       {name} name attr
     | {type} T.type inputtype
     | {attribute} attribute
     ;

  inputtype =
       {texttype} text
     | {radiotype} radio
     | {strtype} stringconst;

  attribute =
       {attr} attr
     | {assign} [left_attr]:attr [right_attr]:attr;

  attr =
```

```
    {id} identifier
  | {str} stringconst
  | {iconst} P.intconst;

intconst =
    {negint} neg_intconst
  |{posint} pos_intconst;

schema = identifier field*;
field = P.type identifier;
variable = P.type identifier+;
identifiers = identifier*;

type =
    {int} int
  | {bool} bool
  | {string} string
  | {void} void
  | {simple} P.type
  | {tuple} identifier;

function = P.type identifier argument* compoundstm;
arguments = argument*;
argument = P.type identifier;
session = identifier compoundstm;

stm =
    {empty}
  | {show} document P.receive?
  | {exit} document
  | {return}
  | {returnexp} exp
  | {if} exp stm
  | {ifelse} exp [then_stm]:stm [else_stm]:stm
  | {while} exp stm
  | {comp} compoundstm
  | {exp} exp;

document =
    {id} identifier
  | {plug} identifier P.plug*;

receive = P.input*;

compoundstm = variable* stm*;

plugs = P.plug*;
plug = identifier exp;

inputs = P.input*;
```

```
input = lvalue identifier;

exp =
    {assign} lvalue [right]:exp
  | {or} [left]:exp [right]:exp
  | {and} [left]:exp [right]:exp
  | {eq} [left]:exp [right]:exp
  | {neq} [left]:exp [right]:exp
  | {lt} [left]:exp [right]:exp
  | {gt} [left]:exp [right]:exp
  | {lteq} [left]:exp [right]:exp
  | {gteq} [left]:exp [right]:exp
  | {plus} [left]:exp [right]:exp
  | {minus} [left]:exp [right]:exp
  | {mult} [left]:exp [right]:exp
  | {div} [left]:exp [right]:exp
  | {mod} [left]:exp [right]:exp
  | {join} [left]:exp [right]:exp
  | {keep} [left]:exp identifier
  | {remove} [left]:exp identifier
  | {keep_many} [left]:exp identifier+
  | {remove_many} [left]:exp identifier+
  | {not} [left]:exp
  | {neg} [left]:exp
  | {default} [left]:exp
  | {lvalue} lvalue
  | {call} identifier exp*
  | {int} P.intconst
  | {true} true
  | {false} false
  | {string} stringconst
  | {tuple} fieldvalue*
  | {paren} exp;

exps = exp*;

lvalue =
    {simple} identifier
  | {qualified} [left]:identifier [right]:identifier;

fieldvalues = fieldvalue*;
fieldvalue = identifier exp;
```

2.2 LEXICAL ANALYSIS USING THE SABLECC TOOL

There are four sections in the specification file that influence the lexer generator of SableCC, the Package, Helpers, States and Tokens sections.

2.2.1 PACKAGE, HELPERS, TOKENS

The WIG keywords we defined in the token sections are: service, const, html, schema, session, show, exit, return, if, else, while, plug, receive, int ,bool, string ,void, tuple, true and false. The HTML part has its own keywords, exempli gracia: “input”. There are tokens defined for arithmetic as well as tuple and boolean operations.

2.2.2 STATES

We identified different states to prevent ‘keyword stealing’. The states are:

1. WIG Scope: all the keywords that are used in wig programs e.g.: service, session, show and so on.

2. HTML Scope: Html scope is anything part of the html language, such as ‘<b>’, ‘<body>’ or meta tags (‘<!-- this is an html comment -->’).

3. Whatever Scope: This is any character excluding ‘<’ and ‘>’.

4. HTML Tag Scope: Html tag scope is for the input tag and it can contain names like ‘type’, ‘name’ or ‘text’ among others, each of which have an attribute.

5. Hole Scope: Plugs use hole variables. The hole variables are in html code in between those 2 tags: ‘<[‘ and ‘]>’.

2.3. PARSING USING SABLECC TOOL

There are four sections in the specification file that influence the parser generator: Package, Tokens, Ignored Tokens and Productions. The ignored tokens we have are blanks, comments and end of line characters. These tokens are recognized by the lexer but not used by the parser.

SableCC 3 has special rules for CST->AST transformations and an AST productions section. That is the main reason we write the productions in our grammar using SableCC 3 style specifications.

A production is transformed in one or several AST productions or tokens. Here is an example of one of our productions:

```
session = T.session identifier l_par r_par compoundstm
  {->New session(identifier, compoundstm)};
```

"T.session", "l_par", "r_par" are tokens while "identifier" and "compoundstm" are other productions. During the parsing time a program, at the reduction phase, the parser will construct nodes for each of the above i.e.: a node for T.session, one for the identifier etc... All the productions map to an AST production in the grammar. SableCC3 will create walker classes which are used to traverse the AST in the future stages.

2.4 ABSTRACT SYNTAX TREES

Construction of WIG abstract syntax trees: The 2 main nodes in the AST are the ones for expressions and statements. A lot of the productions were regrouped under expressions (as 'exp') and statement (as 'stm'). Care was taken not to create unnecessary nodes of different types to avoid absurdly large ASTs.

Only the essential components of the CST are captured in the AST. Let's take functions as an example:

Productions:

```
function{->function} = P.type identifier l_par arguments? r_par compoundstm
        {->New function(P.type, identifier, [arguments.argument], compoundstm)};
```

AST:

```
function = P.type identifier argument* compoundstm;
```

The New token gets rid of the left and right parentheses, which are in the CST just so that the user does not forget to put them when declaring a function, and keeps the type, identifier and body of the function only.

2.5 WEEDING

Weeding cases are presented in Table 2.1.

Table 2.1 Weeding Cases

ID	Case	Reason
WC1	HTML, Schema, Variable, Function and Sessions must not have naming conflicts. We check for global and local variables' conflicts.	Different variables must not have the same name. Otherwise, when the variable name used, it is not possible to know which one to use.
WC2	Fields within schemas must have different names.	We access a tuple element by doing "tuppy.f", where "tuppy" is the tuple name and "f" is a field defined in that tuple's schema. If there are 2 fields named "f", how do we know which one to use? This is why fields have unique names.
WC3	A schema definition cannot be empty	An empty schema is never required.
WC4	The identifier following a tuple declaration must be a valid schema.	A tuple can only be defined via a schema, which specifies the tuple's fields.
WC5	Division by zero must be reported.	Division by zero is undefined.
WC6	Non-void functions must always have a return statement.	When a function is called, if it is expected that it will return a value some type, then the caller is probably assigning that value to some variable, exempli gracia: "tuppy.f = max(1,2);", where max's signature is "int max(int x, int y)". Here, if the function max does not return an int, an error occurs. This is why functions which do not have type void must have a proper return statement.
WC7	A session must always end with an exit statement.	There needs to be a way to end a session.
WC8	Variables cannot be declared with type void.	Only functions can have type void.
WC9	The identifier part of a plug must correspond to an existing hole.	We should not be plugging to a non-existing hole. For the statement "show plug[x=y] someHtml;", there needs to be a hole "<[x]>" in someHtml - otherwise, the plug is useless.
WC10	Receive constructs names' should exist in input tags.	Take this example: "show setup receive[myMsg=userMsg];". There needs to be an input tag with name "userMsg". If there is no input tag with name "userMsg", an error will occur as we have nothing to receive from.
WC11	Chained joins not allowed	For example: "tup1 = tup2 << tup3;" is allowed. However, "tup1 = tup2 << tup3 << tup4;" is not. This is because we did not have time to handle chained joins.
WC12	Some keywords not specified in the grammar are rejected. Those are: Return, list, List. Sessions cannot be named "destroy".	This is because our target language for code generation is PHP and using those words as variable names will cause conflicts.

Here is a code sample in the weeder for checking schemas are legal:

```
public void caseASchema(ASchema node)
{
   String name = node.getIdentifier().getText();
   if(fSchemasNames.contains(name))
   {
      System.out.println("Error: Duplicate schema: " + node.getIdentifier().getText() + "
          at line " + node.getIdentifier().getLine());
      fErrorPresent = true;
   }
   else
   {
      fSchemasNames.add(name);
   }

   Set<String> fieldsNames = new HashSet<String>();
   for(PField field : node.getField())
   {
      AField fieldImpl = (AField) field;
      String memberName = fieldImpl.getIdentifier().toString().trim();
      if(!fieldsNames.contains(memberName))
      {
         fieldsNames.add(memberName);
      }
      else
      {
         System.out.println("Error: Duplicate member " + memberName + " in Schema "
             + name + " declared at line " + node.getIdentifier().getLine());
         fErrorPresent = true;
      }
   }

   if(fieldsNames.isEmpty())
   {
      System.out.println("Error: Definition for schema " + name + " cannot be empty at
          line :" + node.getIdentifier().getLine());
      fErrorPresent = true;
   }
  // …
}
```

2.6 TESTING

Unit testing was used to test whether the AST was being created properly. Concerning the weeder, a significantly large WIG script was made to trigger all possible error messages which could possibly occur during the weeding. When the weeder encounters an anomaly, it does not exit right away, but instead prints out the error, and at the end of the weeding process, if there was at least one error, it exits. A script which the weeder must accept is also used for testing - i.e.: we check whether the weeder does not find errors in totally correct WIG scripts. The weeder reacted perfectly to all tests performed.

We also tried running the weeder on some of the benchmarks provided. It rejected some of them because they had html consts named “Return” or “List” which are not allowed in our language [See WC12]. Apart from that, all the benchmarks were perfectly handled.

3 SYMBOL TABLES

3.1 SCOPE RULES

These are the 5 scopes we identified:

- Service
- HTML
- Schema
- Function
- Session

3.2 SYMBOL TABLE/DATA

A Symbol has a name and a kind. The symbol kinds that occur in WIG programs are: HTML_CONST, INPUT_TAG, SELECT_TAG, HOLE, SCHEMA, VARIABLE, FUNCTION, ARGUMENT, SESSION and FIELD. Usually, a union is used to hold the node value a symbol has. Since we are coding in JAVA we extended the Symbol abstract class into specialized classes: SArgument, SField, SFunction, SHole, SHtml, SInput, SSchema, SSelect, SSession and Svariable.

A class called SymbolTable is used to store the symbols during traversal. Symbol tables existed as specific hashmaps for each scope. The root hash map is the service hash map and it maps identifier strings to their specific symbol. A type symbol is an abstract implementation with sub-classes enforcing type safety. Certain symbols such as functions have their own scope and thus have their own symbol tables. The symbols for the functions contain a reference to their symbol table that is added upon construction of the symbol table. The hash maps themselves exist in a symbol table object that provides specific functionality for our wig implementation.

A Symbol table object has the following fields:

- A hash map of string to symbol.
- Since compound statements have their own specific scope but no unique name to identify it, a specific hash map in each symbol table is used to map compound statement nodes to their specific scope. Special getters and setters use this table for compound statements.
- Another symbol table “fNext” that scopes the current one.

3.3 ALGORITHM

3.3.1 PASS1 : SYMBOL COLLECTION

Very little checking is done during this pass. We leave the heavy work for pass 2. During symbol collection, as its name clearly relates, the ast is traversed and symbols are collected. The only check performed is to check whether variables encountered have already been defined. A service level symbol table and the scoped symbol tables are created (when we change scopes). An example can be seen in the following code:

```
public void inAHtml(AHtml node)
{
   SymbolTable scopedSymbolTable =
          SymbolTable.scopeSymbolTable(fCurrentSymTable);
   fSymbolTables.add(scopedSymbolTable);
   fCurrentSymTable = scopedSymbolTable;
}

public void caseAHtml(AHtml node)
{
          ...
     // collect symbols
     if(SymbolTable.getSymbol(fCurrentSymTable.getNext(), name) != null)
     {
          puts("Error: HTML variable Name " + name + " already defined.");
          System.exit(1);
          }
      else
      {
          SymbolTable.putSymbol(fCurrentSymTable.getNext(), name,
                    SymbolKind.HTML_CONST, node, fCurrentSymTable);
      }
          ...
}

public void outAHtml(AHtml node)
{
   fCurrentSymTable = fCurrentSymTable.getNext();
}
```

3.3.2 PASS2: SYMBOL ANALYZER

The symbol analyzer builds on the symbol tables created during the Symbol Collection. It is initialized with with the symbol table for the service that is passed in to it. The analyzer then traverses through the AST and when it reaches a node that defines its own scope, it pushes it into the current symbol table. Once it exits that node it pops it off (by assigning current symbol table its next value, which is the next table in the hierarchy). Once the traversal reaches either a terminal statement or expression, it pulls all the identifiers and checks if they exist in the current symbol table context, if not it checks if they exist in the hierarchy. The check for duplicate declarations is already done in the collector. If the symbol has not been declared anywhere within a reachable scope, the symbol analyzer declares an error.

Here is a code example showing how we check in a document (in the case where a document is only an identifier) whether there is an html const defined by the document's identifier's name. This is done because we do not want a show statement to try to show an undefined html const.

```
public void caseAIdDocument(AIdDocument node)
{
   String htmlName = node.getIdentifier().getText();
   // check if the html name exists
   if (!SymbolTable.defSymbol(serviceSymbolTable, htmlName))
   {
      System.out.println("Error: Html const '" + htmlName +"' is not defined. Line no:" +
          node.getIdentifier().getLine());
      System.exit(1);
   }
}
```

3.2.3 HANDLING TUPLES DURING SYMBOL TABLE PHASE

Tuples have fields, which are defined when a schema is declared. To handle tuples, we have a class TupleSymbolTable which also has a hash map which maps strings to symbols. It is used to make sure tuple operations are legal. For example: when doing chained remove operations on a tuple, we can determine whether it is legal by calling defsymbol() to check whether the field being removed actually exists in the tuple and then by calling remove() which removes the field.

3.4 TESTING

Various script were written to trigger each an every possible error messages we could have in the Symbol Table checking phase. Another script which is correct was also used for testing whether the symbol table phase does not detect errors when there are none. This phase works perfectly based on our testing.

4 TYPE CHECKING

4.1 TYPES

Describe the types supported by your language.

The types provided by the language are:

- int
- bool
- string
- void
- tuple

4.2 TYPE RULES

These are type rules that are applied to nodes at each relevant visit in the Type Checker. These essentially take the expected types for each rule and programmatically applies their type rule logic. It returns a boolean indicating if the rule works or not. Some rules were combined into one because they possessed the same logic (such as <, >, == etc… for integers were combined into intComparison()).

The type rules in prose follow, some of them accompanied by pieces of code.

4.2.1 ARITHMETIC EXPRESSIONS

4.2.1.1 PLUS

e1 + e2 of type int, then both e1 and e2 are of type int
e1 + e2 of type string, then both e1 and e2 are of type string
e1 + e2 of type string, if e1 is of type int and e2 is of type string
e1 + e2 of type string, if e1 is of type string and e2 is of type int

The TypeChecker has the following piece of code to do this:

```
public void caseAPlusExp(APlusExp node)
  {
     //...
     Type leftNodeType = fTypeTable.getNodeType(node.getLeft());
     Type rightNodeType = fTypeTable.getNodeType(node.getRight());

     if(leftNodeType != null && rightNodeType != null)
     {
        if(TypeRules.intAddition(leftNodeType, rightNodeType))
        {
           fTypeTable.setNodeType(node, Type.INT);
        }
        else if(TypeRules.stringAddition(leftNodeType, rightNodeType))
        {
           fTypeTable.setNodeType(node, Type.STRING);
        }
        else
        {
            // type mismatch error
        }
     }
     //...
  }
```

The above uses the following code in TypeRules class:

```
public static boolean intAddition(Type e1, Type e2)
{
    return e1 == Type.INT && e2 == Type.INT;
}
public static boolean stringAddition(Type e1, Type e2)
{
   return
        (e1 == Type.STRING &&  e2 == Type.STRING) ||
```

```
        (e1 == Type.INT &&  e2 == Type.STRING) ||
        (e1 == Type.STRING &&  e2 == Type.INT);
 }
```

4.2.1.2 MINUS, MULT, DIV, MOD AND NEG

minus: e1 - e2 is of type int, then both e1 and e2 are of type int
mult: e1 * e2 is of type int, then both e1 and e2 are of type int
div: e1 / e2 is of type int, then both e1 and e2 are of type int
mod: e1 % e2 is of type int, then both e1 and e2 are of type int
neg: (- e1) is of type int, then e1 is of type int

4.2.2 COMPARISONS

- lt: e1 < e2 is of type bool, then e1 and e2 are of type int
- gt: e1 > e2 is of type bool, then e1 and e2 are of type int
- lteq: e1 <= e2 is of type bool, then e1 and e2 are of type int
- gteq: e1 >= e2 is of type bool, then e1 and e2 are of type int
- eq: e1 == e2 is of type bool, then e1 and e2 are of the same type
- neq: e1 != e2 is of type bool, then e1 and e2 are of the same type

4.2.3 BOOLEAN OPERATIONS

- not: (! e1) is of type bool, then e1 is of type bool
- or: (e1 || e2) is of type bool, then e1 and e2 must be of type bool
- and: (e1 && e2) is of type bool, then e1 and e2 must be of type bool

4.2.4 ASSIGNMENT

(x = e1) is of type tau: then x is of type tau, e1 is of type sigma and sigma is assignable to tau.

4.2.5 IF/IF-ELSE/WHILE STATEMENTS

If statement: if (e) S: e is of type bool, S is well-formed
If Else statement: if (e) S1 else S2: e must be of type bool, S1 and S2 must be well-formed
While statement: while(e) S: e must be of type bool, S must be well-formed

4.2.6 FUNCTION CALLS/ RETURN EXPRESSION

function call: f(e1,e2,...,en) is of type tau. Check if the types of the e'is match the types of the parameters the function has. Check if f is a function which has return type tau.

Here is code showing how we do this - note that this is not the complete code for type cheking on functions:

In TypeChecker class:

```
public void caseACallExp(ACallExp node)
{
    SFunction symbol = (SFunction) SymbolTable.lookupHierarchy(fCurrentSymbolTable,
                                node.getIdentifier().getText().trim());
    AFunction function = symbol.getFunction();

    Type[] argTypes = getArgumentTypes(function.getArgument());
    Type[] paramTypes;
    {
        List<PExp> copy = new ArrayList<PExp>(node.getExp());
        paramTypes = new Type[copy.size()];

        for(PExp e : copy)
        {
            e.apply(this);
        }

        for(int i=0; i<paramTypes.length; ++i)
        {
            paramTypes[i] = fTypeTable.getNodeType(copy.get(i));
        }
    }

    if(TypeRules.functionCall(argTypes, paramTypes))
    {
        fTypeTable.setNodeType(node, nodeToType(function.getType()));
    }
    else
    {
            // error
    }
    outACallExp(node);
}
```

For return statements:
Check if the return expression and the function return type are the same.

```
public void caseAReturnexpStm(AReturnexpStm node)
{
    inAReturnexpStm(node);
    if(node.getExp() != null)
    {
        node.getExp().apply(this);
    }

    Type nodeType = fTypeTable.getNodeType(node.getExp());

    if(nodeType != null)
    {
        Node parentNode = node.parent();
        while(!(parentNode instanceof AFunction))
        {
            parentNode = parentNode.parent();
        }
        AFunction functionNode = (AFunction) parentNode;
        Type functionType = nodeToType(functionNode.getType());
        if(!TypeRules.returnExpression(functionType, nodeType))
        {
             // type mismatch error
        }
    }
    else
    {
            // error
    }
    outAReturnexpStm(node);
}
```

4.2.7 TUPLES

field value: (e1.x = e2.x) is of type tau, then e1.x is of type tau, e2.x is of type tau

keep: e \+ x is of type T, then e is of type t1, x is of type a, e.x is of type a.

remove: e \- x is of type T, then e is of type t1, x is of type a, e.x is of type a

keep_many: e \+ (x1,x2,x3...) is of type T, then e is of type t1, if xi of type a, then e.xi is of type a

remove_many: e \- (x1,x2,x3...) is of type T, then e is of type t1, if xi of type a, then e.xi is of type a.

join: e1 << e2 is of type tau, then e1 and e2 must be type tuple, e1 and e2 must agree with the attributes they have in common. The resultant tuple of type tau must have a union of
fields from e1 and e2. When two fields in e1 and e2 are the same we check if they have the same basic type. If both the fields are the same we choose the field from e1.

4.3 ALGORITHM

A TypeTable is used during the type checking phase. The class TypeTable is a wrapper around a HashTable which maps Nodes to Types. TypeTable provides methods getNodeType, setNodeType and containsNode, which can be used to get a Node's Type from the TypeTable, set a Node's Type in the TypeTable and check if a Node is already in the TypeTable respectively. TypeTable is used by the TypeChecker.

The TypeChecker implements the DepthFirstAdapter. It has a TypeTable and SymbolTable. The TypeChecker goes through nodes by calling node.apply(this) until a leaf node is reached. The leaf nodes are put in the TypeTable with their respective Types. Then, the TypeChecker checks the Nodes above the leaf nodes and applies the required TypeRules. The SymbolTable is used to check for Symbols not defined in the current scope and get those symbol's nodes. For example: for a function call, we need to get the function's return type and parameters' types so that we can check if the function call is being made with the arguments of the correct types or being assigned to a variable which matches the return type of the function.

4.4 TESTING

The type checker was tested in the same fashion as the symbol table phase. We ran the typechecker on scripts which generated all the possible errors and on scripts with no errors too. The type checker reacted properly to all scripts. The benchmarks were also used for testing and the type checker did well.

5 RESOURCE COMPUTATION

No resource computation was necessary for our compiler

6 CODE GENERATION

6.1 STRATEGY

The target language of our compiler was PHP. Generally, there was a one to one mapping for control flow, expressions and strings. The only complicated part of the strategy occurred in multiple show/receive constructs in a session. That is, wig allows a show/receive to be followed by another show/receive in a session. A further complication is dealing with loop control flows that have show/receives embedded within them.

A wig service in a file called name.wig will generate a single PHP file called name.php. The general execution strategy is shown in Figure 6.1. A PHP session is first started during wig execution. This allows all variables and states to be directly saved to PHP's native session storage.

The essential strategy is to run the entire script execution of the script during multiple show/receives and ensuring the state after a show/receive is restored. Loops are also skipped if they have been already executed unless they are nested.

6.1.1 PHP HELPERS

These are helper functions that assist in executing the wig scripts properly. They primarily consist of functions that help reading and writing to the globals file (which is written into via JSON) and saving/loading the state of the execution for show/receives and loops.

6.1.2 HTML CONSTANT FUNCTIONS

A special function is generated for each HTML constant function that takes holes, the current session and the url for the action as arguments. Holes are injected into each html via PHP's string concatenation operator ".". The HTML is generated into the file and the function exits the execution. Each HTML is wrapped in a form that generates the target action via the url and the action.

6.1.3 SCHEMA ARRAYS

PHP assigns arrays by values. Because of this, schemas are generated into an associative array with default empty values. Thus a tuple declaration is simply converted into a

6.1.4 FUNCTIONS

WIG functions are translated directly to PHP functions.

6.1.5 GLOBALS DECLARATION/INITIALIZATION

Globals are saved in the associative array $_SESSION[“globals”] and are declared and initialized by assigning default values to them. Default values are as follows: int = 0, string = “”, boolean = FALSE. This is important because PHP is dynamically typed therefore variables will not exist if they are not initialized.

6.1.5 SESSIONS

Wig sessions have to be specified in the url. The wig session is not stored in the PHP session because it would would cause collision conflicts if a user is trying to access multiple WIG sessions at once. The required session is stored by $WIG_SESSION = $_GET[“session”]. The session is then selected through a simple control flow.

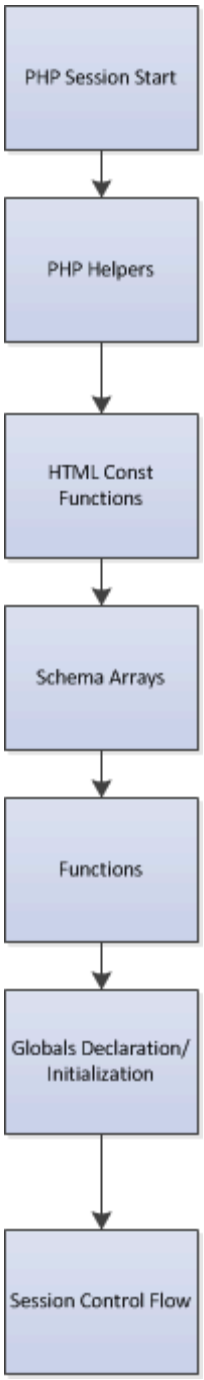

Figure 6.1 General Code Generation Output

6.1.6 HANDLING SUBSEQUENT SHOW/RECEIVE

Figure 6.2 demonstrates the strategy for handling show/receive statements executing serially in a single wig session. Once the session starts it initializes its variables and continues with execution until it hits a show/receive statement. When it hits a show receive it checks to see if it should be skipped, that is, if this the first time the show/receive is being executed in the session, or the first time it is executing in a specific loop iteration.

If the show should be executed, the script sets the current show variable in the PHP session to itself and saves the state upto this point. When execution returns, the script recognizes the specific show was executed so it is skipped and then the state is loaded. The state of globals is freshly loaded and then receive values are assigned (assuming a receive exists). The current show variable in the session is then set to an empty string to allow any future show/receives to execute. The state of the show/receive is also saved.

In the case of returning from a second or subsequent show/receive execution, all previous shows are skipped, their states are loaded and their receives are skipped. This continues until the end of execution or the restart of a loop.

6.1.7 HANDLING SHOW/RECEIVES IN LOOPS

Figure 6.3 demonstrates the loop strategy in In the case of loops, a special variable is set to see if the loop should be skipped or not. This means if a loop is already executed during a session and need not execute again (a re-execution would happen only for nested loops). Once a loop iteration is entered, the state of the loop after the last iteration is saved (apart from the very first iteration). Execution then proceeds as specified. Once the end of an iteration is reached, the state of the execution is saved and all nested loops and show/receive statements inside the loop have their saved states unset. This allows for a fresh execution in the next iteration. Once the loop is done executing, the skip value of the loops state is set to true and the loops execution is saved. If a loop is skipped its state from its completed execution is loaded. Execution then continues as normal and the script exits.

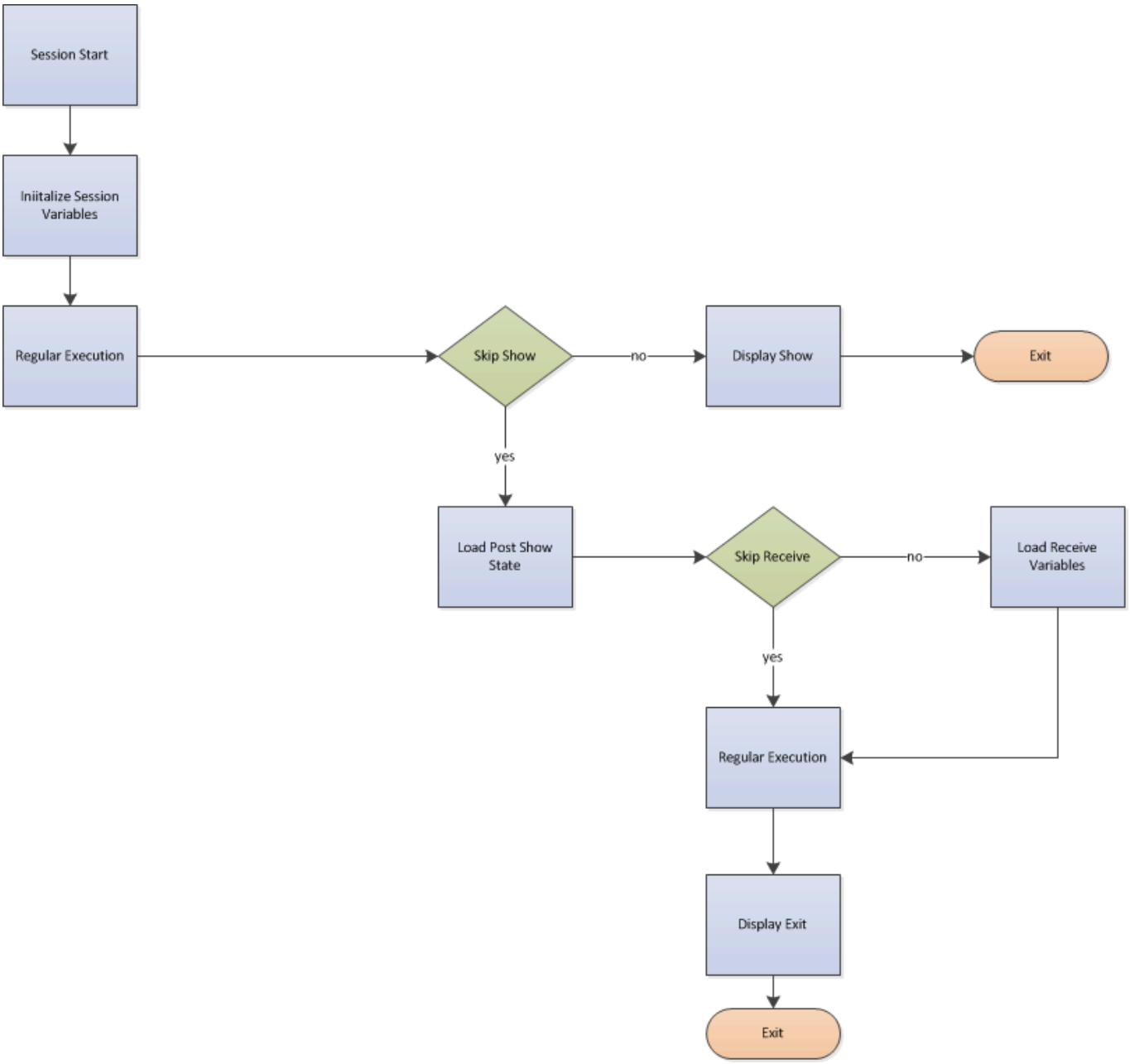

Figure 6.2 Execution of Show/Receive Constructs

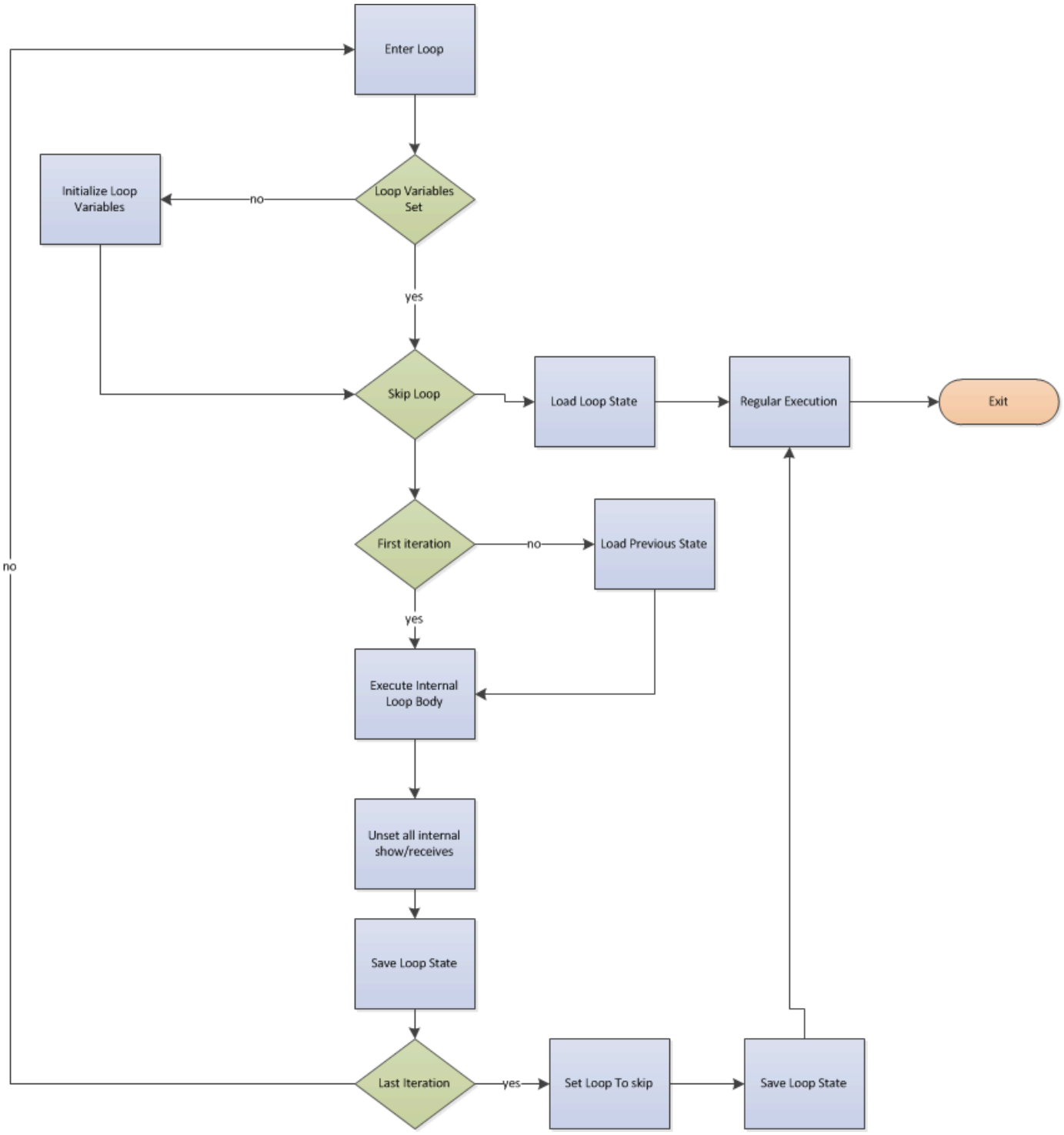

Figure 6.3 Execution of Loops

6.2 CODE TEMPLATES

The code templates for all major code constructs of the wig language are presented in this section.

6.2.1 SERVICE/SESSIONS

A trivial service is shown below.

```
service
{
        const html exitMsg= <html><body></body></html>;
        session One()
        {
                exit exitMsg;
        }

        session Two ()
        {
                exit exitMsg;
        }
}
```

Compiles to:

```
<?php
        session_start();
        function unescapeHTML($html)
        {
          return strtr($html,Array("&lt;"=>"<","&amp;"=>"", "quot;"=>'\'', " " => " ", "&gt" => ">"));
        }

        function array_remove_key($tuppy, $keys) {
          $newTuppy = array();
          foreach($tuppy as $k=>$v)
          {
            if(!in_array($k, $keys))
            {
              $newTuppy[$k] = $v;
            }
          }
          return $newTuppy;
        }

        function createJSON_file($json_file)
        {
                if(!file_exists($json_file))
                {
                        $fileHandle = fopen($json_file, 'w') or die("can't open file");
                        fclose($fileHandle);
                }
        }
```

```
function readGlobals($json_file)
{
   $GLOBALS_FILE = $json_file;
   $line = file_get_contents($GLOBALS_FILE);
   $json_a = json_decode($line, TRUE);
   if(!empty($json_a))
   {
      $_SESSION['globals'] = $json_a['globals'];
   }
}

function writeGlobals($json_file)
{
   $GLOBALS_FILE = $json_file;
   $global_file_handle = fopen($GLOBALS_FILE, 'w');
   $json_a = json_encode($_SESSION);
   fwrite($global_file_handle, $json_a);
   fclose($global_file_handle);
}

function saveLocalsState($label, $currentSession)
{
        if(!isset($_SESSION[$currentSession]["locals_states"]))
        {
                $_SESSION[$currentSession]["locals_states"] = array();
                if(!isset($_SESSION[$currentSession]["locals_states"][$label]))
                {
                        $_SESSION[$currentSession]["locals_states"][$label] = array();
                }
        }
        $_SESSION[$currentSession]["locals_states"][$label]["locals"] = $_SESSION[$currentSession]["locals"];
        $_SESSION[$currentSession]["locals_states"][$label]["globals"] = $_SESSION["globals"];
}

function loadLocalsState($label, $currentSession)
{
        $_SESSION[$currentSession]["locals"] =
$_SESSION[$currentSession]["locals_states"][$label]["locals"];
        $_SESSION["globals"] = $_SESSION[$currentSession]["locals_states"][$label]["globals"];
}
createJSON_file("globals.json");function exitMsg ($holes, $url, $currSessionName){
        $html = "<html><body><form name='".$currSessionName."' action='".$url."' method='get'><br/><input
type='hidden' name='session' value='".$currSessionName."'><input type='submit'
value='Submit'></br></form></body></html>";
        echo unescapeHTML($html);
        exit(0);
}
$globals= array();
$WIG_SESSION = $_GET["session"];
```

```
readGlobals("globals.json");
if (strcmp($WIG_SESSION, "One")==0)
{
        $_SESSION["One"]["locals"]= array();
        writeGlobals("globals.json");
        exitMsg(null, "/~fkhan24/cgi-bin/simple_example.php", "One");

}
else if (strcmp($WIG_SESSION, "Two")==0)
{
        $_SESSION["Two"]["locals"]= array();
        writeGlobals("globals.json");
        exitMsg(null, "/~fkhan24/cgi-bin/simple_example.php", "Two");

}
else if  (strcmp($WIG_SESSION, "destroy")==0)
{
        session_destroy();

}

?>
```

6.2.2 VARIABLE DECLARATION

```
int integer;
string characters;
tuple mySchema tuppy;
```

Compiles To :

In Sessions

```
$_SESSION["One"]["locals"]= array();
$_SESSION["One"]["locals"]["integer"]=0;
$_SESSION["One"]["locals"]["characters"]="";
$_SESSION["One"]["locals"]["tuppy"]=$schema_mySchema;
```

In Globals

```
$globals= array();
$_SESSION["globals"]["integer"]=0;
$_SESSION["globals"]["characters"]="";
$_SESSION["globals"]["tuppy"]=$schema_mySchema;
```

6.2.3 HTML CONSTANT DECLARATION

```
const html exitMsg= <html><body></body></html>;
```

Compiles To:

```
function exitMsg ($holes, $url, $currSessionName){
                $html = "<html><body><form name='".$currSessionName."' action='".$url."' method='get'><br/><input
type='hidden' name='session' value='".$currSessionName."'><input type='submit'
value='Submit'></br></form></body></html>";
                echo unescapeHTML($html);
                exit(0);
        }
```

6.2.4 SCHEMA ARRAY DECLARATION

```
schema mySchema
{
        int my;
        string yours;
}
```

Compiles To:

```
$schema_mySchema= array("my"=>0,"yours"=>"");
```

6.2.5 FUNCTIONS

```
int aFunction()
{
        return 1;
}
```

Compiles To:

```
function aFunction()
{
        return 1;
}
```

6.2.6 EXPRESSIONS

```
1 + 2*2 - 3 / 4;
```

Compiles To:

```
1 + 2 * 2-3 / 4;
```

6.2.7 STATEMENTS

```
number = 1 + 2*2 - 3 / 4 + aFunction();
```

Compiles TO:

```
$_SESSION["One"]["locals"]["number"] = 1 + 2 * 2-3 / 4 + aFunction();
```

6.2.8 IF/ELSE CONTROL FLOW

```
if(1 < 0){}
else{}
```

Compiles To :

```
if(1 < 0){}
else{}
```

6.2.9 SHOW/RECEIVE

```
show setup receive[userMsg=userMsg];
show plug rReturn [allMsg=messages] receive[another=another];
```

COMPILES TO:

```
if (!isset($_SESSION["Calculate"]['currShow']))
{
	$_SESSION["Calculate"]['currShow'] = "";

}
if (strcmp($_SESSION["Calculate"]['currShow'],"") == 0)
{
	$_SESSION["Calculate"]['currShow'] = "show2";
	saveLocalsState("show2","Calculate");
	writeGlobals("globals.json");
	setup(null, "/~fkhan24/cgi-bin/wall.php", "Calculate");

}
loadLocalsState("show2","Calculate");
if (strcmp($_SESSION["Calculate"]['currShow'],'show2') == 0)
{
	readGlobals("globals.json");
	if(isset($_GET['userMsg']))
	{
		$_SESSION["Calculate"]['locals']['userMsg'] = $_GET['userMsg'];
	}
	$_SESSION["Calculate"]['currShow'] = "";
}

if (!isset($_SESSION["Calculate"]['currShow']))
{
	$_SESSION["Calculate"]['currShow'] = "";
}
if (strcmp($_SESSION["Calculate"]['currShow'],"") == 0)
{
	$_SESSION["Calculate"]['currShow'] = "show4";
	saveLocalsState("show4","Calculate");
	writeGlobals("globals.json");
	rReturn(array("allMsg" => $_SESSION["globals"]["messages"]), "/~fkhan24/cgi-bin/wall.php", "Calculate");
```

```
}
loadLocalsState("show4","Calculate");
if (strcmp($_SESSION["Calculate"]['currShow'],'show4') == 0)
{
	readGlobals("globals.json");
	if(isset($_GET['another']))
	{
		$_SESSION["Calculate"]['locals']['another'] = $_GET['another'];
	}
	$_SESSION["Calculate"]['currShow'] = "";
}
```

6.2.10 WHILE LOOP CONTROL FLOW

```
while (another != "no")
{
	show setup receive[userMsg=userMsg];

	show plug rReturn [allMsg=messages] receive[another=another];
}
```

Compiles To :

```
if(!(isset($_SESSION["Calculate"]["locals_states"]["loop1"]) && $_SESSION["Calculate"]["locals_states"]["loop1"]["skip"]))
{
	while($_SESSION["Calculate"]["locals"]["another"] != "no" )
	{
		if(isset($_SESSION["Calculate"]["locals_states"]["loop1"]["first"]) &&
!$_SESSION["Calculate"]["locals_states"]["loop1"]["first"])
		{
			loadLocalsState("loop1", "Calculate");

		}

		// SHOW/RECEIVE PART OMITTED, SEE PREVIOUS SECTION FOR OUTPUT

		unset($_SESSION["Calculate"]["locals_states"]["show2"]);
		unset($_SESSION["Calculate"]["locals_states"]["show4"]);
		saveLocalsState("loop1", "Calculate");

	}
	$_SESSION["Calculate"]["locals_states"]["loop1"]["skip"]=TRUE;saveLocalsState("loop1","Calculate");

}
else
{
	loadLocalsState("loop1","Calculate");
```

```
}
```

6.2.11 TUPLE OPERATIONS

PHP's native array operations provided an easy methodology towards handling the tuple operations.

```
schema scheme
{
        int x;
        int y;
        int z;
}

schema scheme1
{
        bool a;
        string b;
}

tuple scheme tuppy;
tuple scheme tuppy1;
tuple scheme1 tuppy2;
tuple scheme1 tuppy3;
tuple scheme tuppy4;
tuple scheme tuppy5;
tuple scheme tuppy6;

tuppy = tuppy1 \+ y;
tuppy2 = tuppy3 \- a;
tuppy2 = tuppy3 << tuppy1;
tuppy4 = tuppy5 \- (x,z);
tuppy5 = tuppy6 \+ (x,y);
```

COMPILES TO:

```
$schema_scheme= array("x"=>0,"y"=>0,"z"=>0);
$schema_scheme1= array("a"=>FALSE,"b"=>"");

$_SESSION["globals"]["tuppy1"]=$schema_scheme;
$_SESSION["globals"]["tuppy2"]=$schema_scheme1;
$_SESSION["globals"]["tuppy3"]=$schema_scheme1;
$_SESSION["globals"]["tuppy4"]=$schema_scheme;
$_SESSION["globals"]["tuppy5"]=$schema_scheme;
$_SESSION["globals"]["tuppy6"]=$schema_scheme;

$_SESSION["globals"]["tuppy"] = array("y" => $_SESSION["globals"]["tuppy1"]["y"]);
$_SESSION["globals"]["tuppy2"] = array_remove_key($_SESSION["globals"]["tuppy3"], array("a"));
$_SESSION["globals"]["tuppy2"] = array_merge($_SESSION["globals"]["tuppy3"], $_SESSION["globals"]["tuppy1"]);
```

```
$_SESSION["globals"]["tuppy4"] = array_remove_key($_SESSION["globals"]["tuppy5"], array("x", "z");
$_SESSION["globals"]["tuppy5"] = array("x" => $_SESSION["globals"]["tuppy6"]["x"], "y" =>
$_SESSION["globals"]["tuppy6"]["y"]);
```

6.3 ALGORITHM

Code generation is done in a class called Emitter. It extends depth first adapter and is called via the emit method. Each construct node that is visited has its specific PHP equivalent then outputted. The PHP equivalents are shown in 6.2. Most specific cases of code generation have already been explained in the strategy part in 6.1.

Since multiple execution states need to be saved, a consistent mechanism for doing so was designed. There are loop and show functions that generate a unique label each time they are called. These are used to then assign state saving and loading in the locals_state array in the session. All of these labels are saved in a label map so that they can be accessed throughout the code and by the LoopLabelCollector.

The LoopLabelCollector is another implementation of depth first adapter used when while loops are emitted. The label collector detects all nested loops and show receives within one level of nesting. It collects all the specific labels for these constructs from the label map and constructs a list. Code is then emitted to unset these after each while loop iteration is complete.

There is a function that detects if loops are within a function. Since show/receives are not permitted within a function, a simple translation from wig while loops to PHP while loops is done without any of the handling done in sessions.

PHP helpers are saved in the phphelpers.txt file in the src folder. These helpers are loaded by the PHPHelper class and are one of the first things written to the script. They are consistent throughout all wig scripts.

The Emitter also uses the type table obtained from the type checking phase. This is done because the types of some nodes are required in some cases e.g.: when adding 2 strings, a ‘.’ is used, whereas for ints, a “+”. So we need to know the type of the nodes in APlusExp.

6.4 RUNTIME SYSTEM

A web server with a PHP module is required. Testing for this phase was done with the LAMP (and in some cases MAMP) stack. The php scripts need to be executable by groups trying to access them. And before code is run a globals.json file must be created in the same directory as the php script.

6.5 SAMPLE CODE

The compilation of tiny.wig (http://www.sable.mcgill.ca/~hendren/520/WIG99/tiny.wig.txt) is shown below:

```
<?php
        session_start();
        function unescapeHTML($html)
        {
          return strtr($html,Array("&lt;"=>"<","&amp;"=>"", "quot;"=>'\'', " " => " ", "&gt" => ">"));
        }

        function array_remove_key($tuppy, $keys) {
          $newTuppy = array();
          foreach($tuppy as $k=>$v)
          {
            if(!in_array($k, $keys))
            {
              $newTuppy[$k] = $v;
            }
          }
          return $newTuppy;
        }

        function createJSON_file($json_file)
        {
                if(!file_exists($json_file))
                {
                        $fileHandle = fopen($json_file, 'w') or die("can't open file");
                        fclose($fileHandle);
                }
        }

        function readGlobals($json_file)
        {
          $GLOBALS_FILE = $json_file;
          $line = file_get_contents($GLOBALS_FILE);
          $json_a = json_decode($line, TRUE);
          if(!empty($json_a))
          {
            $_SESSION['globals'] = $json_a['globals'];
          }
        }

        function writeGlobals($json_file)
        {
          $GLOBALS_FILE = $json_file;
          $global_file_handle = fopen($GLOBALS_FILE, 'w');
          $json_a = json_encode($_SESSION);
          fwrite($global_file_handle, $json_a);
          fclose($global_file_handle);
        }
```

```
function saveLocalsState($label, $currentSession)
{
	if(!isset($_SESSION[$currentSession]["locals_states"]))
	{
		$_SESSION[$currentSession]["locals_states"] = array();
		if(!isset($_SESSION[$currentSession]["locals_states"][$label]))
		{
			$_SESSION[$currentSession]["locals_states"][$label] = array();
		}
	}
	$_SESSION[$currentSession]["locals_states"][$label]["locals"] = $_SESSION[$currentSession]["locals"];
	$_SESSION[$currentSession]["locals_states"][$label]["globals"] = $_SESSION["globals"];
}

function loadLocalsState($label, $currentSession)
{
	$_SESSION[$currentSession]["locals"] =
$_SESSION[$currentSession]["locals_states"][$label]["locals"];
	$_SESSION["globals"] = $_SESSION[$currentSession]["locals_states"][$label]["globals"];
}
createJSON_file("globals.json");function Welcome ($holes, $url, $currSessionName){
	$html = "<html><body><form name='".$currSessionName."' action='".$url."'
method='get'>Welcome!<br/><input type='hidden' name='session' value='".$currSessionName."'><input type='submit'
value='Submit'></br></form></body></html>";
	echo unescapeHTML($html);
	exit(0);
}
function Pledge ($holes, $url, $currSessionName){
	$html = "<html><body><form name='".$currSessionName."' action='".$url."' method='get'>How much do
you want to contribute?<input  name=&amp;quot;contribution&amp;quot; type=&amp;quot;text&amp;quot; size=4
/><br/><input type='hidden' name='session' value='".$currSessionName."'><input type='submit'
value='Submit'></br></form></body></html>";
	echo unescapeHTML($html);
	exit(0);
}
function Total ($holes, $url, $currSessionName){
	$html = "<html><body><form name='".$currSessionName."' action='".$url."' method='get'>The total is
now".$holes["total"].".<br/><input type='hidden' name='session' value='".$currSessionName."'><input type='submit'
value='Submit'></br></form></body></html>";
	echo unescapeHTML($html);
	exit(0);
}
$globals= array();$_SESSION["globals"]["amount"]=0;

$WIG_SESSION = $_GET["session"];
readGlobals("globals.json");
if (strcmp($WIG_SESSION, "Contribute")==0)
{
	$_SESSION["Contribute"]["locals"]= array();
	$_SESSION["Contribute"]["locals"]["i"]=0;
```

```
                $_SESSION["Contribute"]["locals"]["i"] = 87;
                ;if (!isset($_SESSION["Contribute"]['currShow'])){
                        $_SESSION["Contribute"]['currShow'] = "";

                }
                if (strcmp($_SESSION["Contribute"]['currShow'],"") == 0)
                {
                        $_SESSION["Contribute"]['currShow'] = "show2";
                        saveLocalsState("show2","Contribute");
                        writeGlobals("globals.json");
                        Welcome(null, "/~fkhan24/cgi-bin/tiny.php", "Contribute");

                }
                loadLocalsState("show2","Contribute");
                if (strcmp($_SESSION["Contribute"]['currShow'],'show2') == 0)
                {
                        readGlobals("globals.json");$_SESSION["Contribute"]['currShow'] = "";

                }
                if (!isset($_SESSION["Contribute"]['currShow'])){
                        $_SESSION["Contribute"]['currShow'] = "";

                }
                if (strcmp($_SESSION["Contribute"]['currShow'],"") == 0)
                {
                        $_SESSION["Contribute"]['currShow'] = "show4";
                        saveLocalsState("show4","Contribute");
                        writeGlobals("globals.json");
                        Pledge(null, "/~fkhan24/cgi-bin/tiny.php", "Contribute");

                }
                loadLocalsState("show4","Contribute");
                if (strcmp($_SESSION["Contribute"]['currShow'],'show4') == 0)
                {
                        readGlobals("globals.json");
                        if(isset($_GET['contribution']))
                        {
                                $_SESSION["Contribute"]['locals']['i'] = intval($_GET['contribution']);
                        }
                        $_SESSION["Contribute"]['currShow'] = "";

                }
                $_SESSION["globals"]["amount"] = $_SESSION["globals"]["amount"] +
$_SESSION["Contribute"]["locals"]["i"];
                ;writeGlobals("globals.json");
                Total(array("total" => $_SESSION["globals"]["amount"]), "/~fkhan24/cgi-bin/tiny.php", "Contribute");

        }
        else if  (strcmp($WIG_SESSION, "destroy")==0)
        {
```

```
            session_destroy();

        }

?>
```

6.6 TESTING

The code works on almost all of the 2009 group benchmarks. The benchmarks that did not work were group 1 and group 6. Group 1 did not work because their grammar differed from ours. Group 6's implementation of Mastermind was also rejected because the implementation of selects in our compiler was not handled properly.

The benchmarks for Group 2, 3, 4, 5 and 7 all produced scripts that executed. Some work was required such as renaming "end" and "list" to another identifier because they are keywords in PHP. Most of the scripts executed as expected. Some scripts produced non-fatal errors that did not disrupt the execution flow.

After a few changes in group 1's wig code, the benchmark for group 1 also worked. Group 6's implementation of Tic-Tac-Toe worked.

There were a few redundancy problems that were not fixed due to time constraints. Semi-colons were being printed twice for statements, although this does not cause any execution problems in PHP it makes the output syntactically harder to read.

7 AVAILABILITY AND GROUP DYNAMICS

7.1 MANUAL

COMPILE, INSTALL and RUN WIG programs:

If you want to compile “wig_file.wig”:

- compile: java wig.compiler.Compiler wig_file.wig -cg -up <URL> -ph <PUBLIC_HTML_PATH>
  where <URL> is your base url e.g: mine is "/~dbhage/"
  and <PUBLIC_HTML_PATH> is the path to your public_html,
  e.g.: mine is "/home/2010/dbhage/public_html/"

  IMPORTANT: please do the -up part before the -ph

This will generate “wig_file.php” in your public_html.

- run: In your browser, paste:
  “www.cs.mcgill.ca/~username/wig_file.php?session=SESSION_NAME"
  where SESSION_NAME is the session you want to start at.
  where “username” needs to be replaced by your SOCS username

7.2 DEMO SITE

To reset any of the following benchmarks please change the session argument to “destroy”.
2009 Benchmark Demos:
Group-1:
http://cs.mcgill.ca/~fkhan24/cgi-bin/cs520/group-1//wigg.php?session=gist

Group-2:
http://cs.mcgill.ca/~fkhan24/cgi-bin/cs520/group-2/chat.php?session=Chat
http://cs.mcgill.ca/~fkhan24/cgi-bin/cs520/group-2/emailer.php?session=Emailer [2 lines of the output had to be modified because of url and php keyword issues]

Group-3:
http://cs.mcgill.ca/~fkhan24/cgi-bin/cs520/group-3/wigshop.php?session=Welcome

Group-4:
http://cs.mcgill.ca/~fkhan24/cgi-bin/cs520/group-4/battle.php?session=Choose
http://cs.mcgill.ca/~fkhan24/cgi-bin/cs520/group-4/battle.php?session=Fight
[End had to be changed to end2 and </input> had to be removed from wig file]

Group-5:
http://cs.mcgill.ca/~fkhan24/cgi-bin/cs520/group-5/fatchecker.php?session=Fat

Group-6:
http://cs.mcgill.ca/~fkhan24/cgi-bin/cs520/group-6/TicTacToe.php?session=Move

Group-7:
http://cs.mcgill.ca/~fkhan24/cgi-bin/cs520/group-7/mastermind.php?session=NewGame
http://cs.mcgill.ca/~fkhan24/cgi-bin/cs520/group-7/mastermind.php?session=Mastermind

The entire public html folder is available at:
http://cs.mcgill.ca/~fkhan24/wig_compiler_public_html.zip

7.3 DIVISION OF GROUP DUTIES

All work was done concurrently with all 3 group members.

8 CONCLUSIONS AND FUTURE WORK

8.1 CONCLUSIONS

This report reflects the work of group-h in COMP520 for the year 2012. The report first describes the state of the compiler upon its final submission in section 1.

The grammar used for this compiler is then presented in Section 2. A discussion on the implementation of Lexical Analysis and Parsing through SableCC's automated Scanner and Parser is also given. The abstract syntax tree and the weeding process are also discussed in this section.

Section 3 introduces the Symbol Table strategy. It explains the symbol table data structure used. It also describes the processes by which the symbols are collected and analyzed. Section 4 presents a similar discussion on type checking and also presents the specific type rules that were used for this compiler. Section 5 of the report is empty for this report because no resource calculations were necessary from our compiler.

Section 6 describes the code generation process. The general strategy and the strategy towards handling the complex control flow of show/receive statements is given. Code templates and sample code generation for the tiny.wig file is also presented. An output was generated for almost all the benchmarks for the groups from 2009. Some of the wig files had to be changed to conform to PHP. One's output php required a few conflicts to be removed. Overall they perform as expected. Group 6 Mastermind was the only one that could not be generated properly. Section 7 presents installation instructions and links to the demos of the generated benchmarks.

The compiler has a few small errors but it generates operable code the majority of the time. Run time errors are usually due to incorrect url arguments or PHP keyword conflicts.

8.2 FUTURE WORK

The wig language needs a construct to handle scripts. Also a default method of submission was used for our compiler that generates a submit button. A method to override this default would be useful.

8.3 COURSE IMPROVEMENTS

The benchmarks do not contain examples of tuple operations such as keep, keep many etc... It would be nice to have that.

The documentation for SableCC is very limited. Improved documentation would help significantly. Also a proper specification for the WIG language would be greatly appreciated. Some ambiguity is understandable for freedom in design, however there are too many variations and a specific standard for the majority of the language would have been very useful.

We did not get any feedback from the TA for the WIG compiler deliverables. It would have been very useful to know what the TA thinks of the deliverables, so that improvements could be made for the final submission.

8.4 GOODBYE

Amine Sahibi will be doing his undergraduate thesis with the Compilers Group.

Faiz Khan will be developing an HTML5 Canvas Templating/Animation language to help code web applications using jastAdd.

Dwijesh Bhageerutty is thinking of joining an open source compiler group.

8.5 REFERENCES

1. SABLECC, AN OBJECT-ORIENTED COMPILER FRAMEWORK by Etienne Gagnon, School of Computer Science, McGill University, Montreal
[http://sablecc.sourceforge.net/downloads/thesis.pdf]

2. List of Keywords in PHP: http://php.net/manual/en/reserved.keywords.php