Wig Compiler Report

Sagar Sen
Ximeng Sun

November 28, 2005

Contents

List of Figures

List of Tables

1 Introduction

1.1 Clarifications

1.2 Restrictions

So far, we don't support 'show' statement in 'function' scope; any 'return' should only appear in session parts. We return errors when we have these situations.

1.3 Extensions

No extensions

1.4 Implementation Status

We've implemented most basic features of WIG especially for all operations of tuples (initialization, keep, remove, join, compare, assign).

We designed and implemented an Embedded Virtual Machine to run generated CGI Python Scripts. This way we can have complete control over the program execution. Based-on some nice feature of Python, the EVM is implemented in a very light way.

However, our implementation has the above (see Restrictions) because we didn't have time to finish all work.

2 Parsing and Abstract Syntax Trees

2.1 The Grammar

Give the full grammar for your version of the WIG language by listing your `bison` input with all actions removed.

```
/*******************************************************************
*  Group -03 2005 (Sagar and Ximeng) WIG SableccV3 Grammar          *
*******************************************************************/

Package wig;

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

  all                   = [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;
  notdash               = [all - '-'];
  notmeta_stmt          = [notdash -'>'];
  notquote              = [all-'"'];
  notstar               = [all-'*'];
  notstar_backslash     = [notstar -'/'];
  not_cr_lf             = [all - [lf+cr]];
  eol                   = lf|cr|cr lf;
  escapequote           = '\"';
  meta_start            = '<!--';
  meta_end              = '-->';
  meta_stmt             = meta_start notdash* '-'+ (notmeta_stmt notdash* '-'+)*  meta_end;
  string_const          = '"' (notquote* escapequote* notquote*)* '"';
  sl_comment            = '//' not_cr_lf* eol;
  ml_comment            = '/*' notstar* '*'+(notstar_backslash notstar* '*' +)* '/';
  comment               = sl_comment|ml_comment;
  whatever_stmt         = [[all-'<']-'>']*; //anything without < and >

/*******************************************************************
 * States                                                          *
 *******************************************************************/
States
 wig_block,html_code,whatever_text,form_input;


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

  /*********************
   * Keywords          *
   *********************/
    {wig_block} service = 'service';
    {wig_block} const = 'const';
    {wig_block} html = 'html';
    {wig_block->html_code,html_code} html_tag_start = '<html>';
    {html_code->wig_block,whatever_text->html_code,wig_block} html_tag_end = '</html>';
    {html_code->form_input} input = 'input';
    {html_code,form_input,wig_block}  posint_const = '0' | [digit-'0'] digit*;
    {html_code}  negint_const = '-' [digit-'0']  digit*;
    {html_code} select = 'select';
    {form_input} type = 'type';
    {form_input} name = 'name';
    {form_input} text = 'text';
    {form_input} radio = 'radio';
    {wig_block} schema = 'schema';
    {wig_block} session = 'session';
    {wig_block} show = 'show';
    {wig_block} exit = 'exit';
    {wig_block} return = 'return';
    {wig_block} if = 'if';
    {wig_block} else = 'else';
```

```
  {wig_block} while = 'while';
  {wig_block} plug = 'plug';
  {wig_block} receive = 'receive';
  {wig_block} int = 'int';
  {wig_block} bool = 'bool';
  {wig_block} string = 'string';
  {wig_block} void = 'void';
  {wig_block} tuple = 'tuple';
  {wig_block} true = 'true';
  {wig_block} false = 'false';
/*********************
 * Operators         *
 *********************/
  {wig_block} l_brace = '{';
  {wig_block} r_brace = '}';
  {wig_block,html_code,form_input} assign = '=';
  {wig_block, html_code->wig_block} semicolon = ';';
  {wig_block,whatever_text->html_code,html_code} lt = '<';
  {wig_block,html_code->whatever_text,form_input->whatever_text} gt = '>';
  {whatever_text->html_code,html_code} lt_slash = '</';
  {whatever_text->html_code,html_code} lt_bracket = '<[';
  {html_code->whatever_text} gt_bracket = ']>';
  {wig_block} comment=comment;
  {wig_block} l_par = '(';
  {wig_block} r_par = ')';
  {wig_block} l_bracket = '[';
  {wig_block} r_bracket = ']';
  {wig_block} comma = ',';
  {wig_block} keep = '\+';
  {wig_block} remove = '\-';
  {wig_block} join = '<<';
  {wig_block} eq = '==';
  {wig_block} neq = '!=';
  {wig_block} lteq = '<=';
  {wig_block} gteq = '>=';
  {wig_block} not = '!';
  {wig_block} minus = '-';
  {wig_block} plus = '+';
  {wig_block} mult = '*';
  {wig_block} div = '/';
  {wig_block} mod = '%';
  {wig_block} and = '&&';
  {wig_block} or = '||';
  {wig_block} dot = '.';
  {wig_block,html_code,form_input} eol = eol;
  {wig_block,html_code,form_input} blank = (tab|' '|eol)*;
/*********************
 * Literals          *
 *********************/
  {wig_block,html_code,form_input} identifier = letter_or_us (letter_or_us|digit)*; // usual identifiers
  {wig_block, form_input, html_code} stringconst = string_const ; // usual string constants
  {html_code} meta = meta_stmt; // any string of the form <!-- ... -->
  {whatever_text} whatever = whatever_stmt; // 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} intconst {->New attr.iconst(intconst)}
```

```
;
intconst{->intconst}  =
   {negint}   negint_const {->New intconst.negint(negint_const)}
 | {posint}   posint_const {->New intconst.posint(posint_const)}
;
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)}
 | {compstm} compoundstm  {-> New stm.compstm(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)}
 | {compstm} compoundstm {-> New stm.compstm(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}    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)}
;
/*******************************************************************
 * AST Productions                                                 *
 *******************************************************************/
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}      intconst
;
intconst  = {negint}   negint_const
           |{posint}  posint_const
;
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
           | {compstm} 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}         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

We used SableCC to specify and generate the lexical analyzer. SableCC has the convenience of allowing the implementor to provide both the tokens and the grammar in the same file. Additionally it allows the use of helpers that adds clarity to writing the lexer and parser specification.

The tokens specified for our scanner are categorized in 4 different groups. They are :

1. Keywords: Keywords are the tokens that are part of the WIG language. By definition keywords are reserved and hence cannot be used as a variable or a function name. This check has been implemented in our weeding phase. Please check section 2.6 for details.

2. Operators: The operators consist of arithmetic, relational, boolean, and paranthetic operators.

3. Literals: Literals are integer constants, string constants, identifiers, meta-statements, and whatever statements (this appears within html tags)

4. Ignored Tokens: The ignored tokens are blank, comments, and eol.

We have also used states in SableCC to further limit tokens to only certain sections of code. The states that we use are:

a. wig_block: The wig_block state implies that the token can appear in any part of the wig code except for states that it does not belong to. Egs.: l_par, r_par, const etc.

b. html_code: The html_code state is activated when the parser enters the html body part of a wig program. Keywords specific to this area of the code are recognized here. For instance lt_slash, lt_bracket are identified only in the html_code state. If these tokens appear in other states like the wig_block then

c. whatever_text: In the whatever_text state the parser only reads from

d. form_input: In the form_input state the parse reads the parameters specified in the form tag. Hence it reads parameters of the form type = 'type', name = 'name', text = 'text', radio = 'radio',identifier literal etc. It does not expect other tokens like service or session to appear in this state. Once the parser reads the html tag to exit the form_input state, it returns to the html mode.

2.3 Parsing using `SableCC` Tool

We encoded the productions in our grammar using SableCC v3 style specifications. The main reason for using SableCC v3 is due to the fact that it can generate the abstract syntax tree based on rules specified within the grammar (to transform the concrete syntax tree to an abstract syntax tree). It also generates walker classes to traverse the abstract syntax tree.

All the concrete syntax productions have a mapping to abstract syntax productions in the grammar. SableCC automatically generates the abstract syntax tree for the later stages of the compiler. SableCC first creates a concrete syntax tree for the input wig file. Given the rules of conversion from concrete syntax to abstract syntax, SableCC generates an abstract syntax tree. Future operations like creating symbol tables, type checking, weeding, and code generation use the walker classes created by SableCC to traverse the abstract syntax tree.

2.4 Abstract Syntax Trees

The abstract syntax tree (AST) captures the essential components of the concrete syntax tree (CST). The AST gets rid of syntactical redundancies present in the concrete syntax tree.

For example consider this conversion:

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

Here the production for html is given for the concrete syntax tree. We take a note that for future use of this statement only a couple of tokens or productions are essential. The New statement gets rid of the extra tokens and extracts identifier and [htmlbody] (this is used to represent a list of statements, html code in this case) only.

The corresponding AST will only have the following production:

```
html = identifier htmlbody*;
```

Concrete syntax is usually required to constrain the programmer from making mistakes and for making code more readable. However, the AST is the essential part of the CST that is required for all future stages discussed in this report.

There are four types of transformations that are possible while converting a concrete syntax tree to an abstract syntax tree.

1. Getting an already existing element :: (ident)
2. New alternative (New production[.nameofalternative]) :: creation of a new node

```
Eg:
CST
For type 1 and 2.
arguments{->argument*} =
   {one}  argument    {->[argument]}
```

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

3. List creation ([elem1 elem2 ...]) :: creation of a homogeneous list of terms

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

4. Elimination (Null) :: used in general to replace an element or two to eliminate the effect of another one.

Elimination was used extensively for desugaring.

```
Eg: Here we get rid of the semicolon
variable{->variable} =
   P.type identifiers semicolon
   {->New variable(P.type, [identifiers.identifier])}
;
```

5. Empty transformation :: used in general to get rid of the entire subtree.

Not used in our grammar.

For a complete specification of the abstract syntax tree please refer to section 2.

2.5 Desugaring

Syntatic sugar in WIG does not affect the expressiveness of the language. It is used to make the language more human readable. As mentioned in an example in the previous section, we got rid of punctuations, assignment operators etc. to extract only the essential parts for the abstract syntax.

2.6 Weeding

Do you weed unwanted parse trees? How and why?

2.7 Testing

We tested our grammar for scanning and parsing for all the benchmarks for 2005. To verify that we obtained the correct abstract syntax tree we used the ASTDisplay.java program to display the abstract syntax tree. We also coded a pretty printer to check if pretty(parse(X)) = pretty(parse(pretty(parse(X)))). Where X is the abstract syntax tree generated using our grammar. This pretty printer successfully produced the same code from the abstract syntax tree that we created for all the programs in the benchmark.

3 Symbol Tables

It is important to remember declarations of variables and functions to detect inconsitencies and misuses during the type checking phase. The task of the symbol table is to maintain records of variable and function declarations so that the type checker can use this knowledge to verify the types of each datastructure. A symbol table is a compile-time data structure. It's not used during run time by statically typed languages.

Formally, a symbol table maps names into declarations (called attributes), such as mapping the variable name x to its type int. More specifically, a symbol table stores:

* for each type name, its type definition

* for each variable name, its type. If the variable is an array, it also stores dimension information. It may also store storage class, offset in activation record etc.

* for each constant name, its type and value.

* for each function and procedure, its formal parameter list and its output type. Each formal parameter must have name, type, type of passing (by-reference or by-value), etc.

3.1 Scope Rules

The scope of a name (variable names, data structure names, procedure names) is the part of the program within which the name can be used. We identified 5 different scopes for our symbol table implementation.

*Service

*HTML

*Schema

*Function

*Session

As and when the walker enters these scopes it creates a local symbol table. When the walker exits the scope the symbol table is unscoped. The following sections explain in detail the process of generating symbol tables.

3.2 Symbol Data

The symbol table class contains the following data structures:

* An **integer** that stores the maximum size of hashTable for a particular scope. In our case it is 317. The reason for choosing this number is based on experiments done on several programs. If the hashtable size is larger than this number it normally indicates extreme coding technique and hence the programmer needs to rewrite his code for more elegance.

* A **hashtable** that stores the local variable name and the corresponding Symbol object.

We digress for a moment here to explain what the **Symbol** object is. The Symbol class contains three data-structures.

- A name of type String.

- A reference to a Node from the abstract syntax tree as generated by SableCC

- A Symbolkind variable (a String type) is used to indicate the nature (in terms of scope and location) of the variable or scope (function, session, or service) name. The symbols have the following kinds: html code, schema, service name, service variable, function name, function variable, session name, session variable, function argument, tuple field, and a hole in the html code. Note that variables are different in different scopes.

A constructor is defined to allow setting of these variables in the Symbol object from outside the scope of the class. Now we return to our explanation of the SymbolTable data-structure.

* A reference to the **parent SymbolTable**. When the walker enters a new scope. It creates a new symbol table and marks the parent of that symbol table to *this* (where *this* is the previous scope of the walker. The unscope member function returns this parent of the current symbol table. This method is called when the walker is out of a particular scope. At this point it needs access to the outer symbol table. Hence, unscope provides the walker with exactly that. In figure 3.2 we show the important scoped hashtable that is generated during the symbol table generation phase.

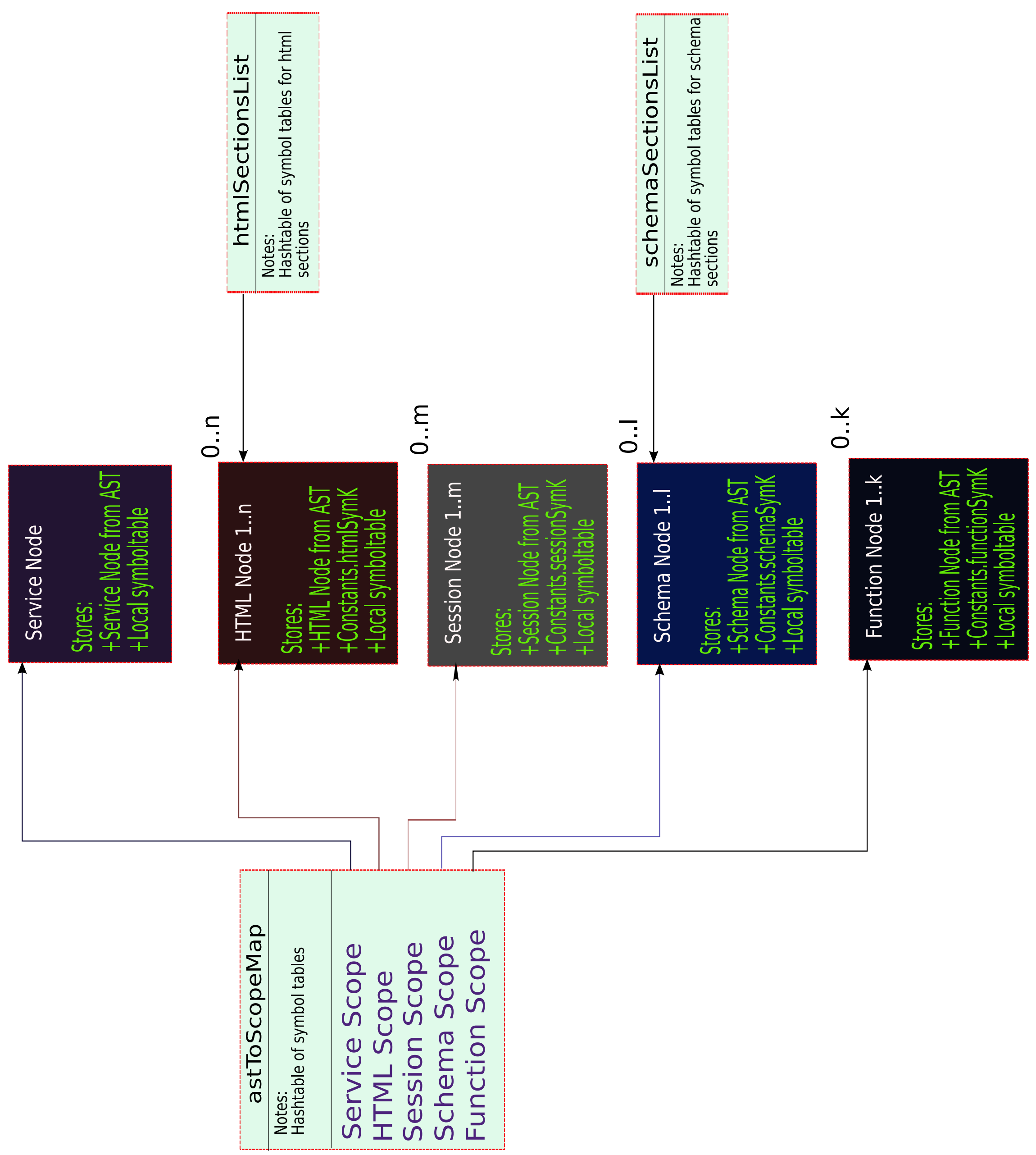

Figure 1: The astToScope, htmlSections, and schemaSections hashtables that store the scoped symbol tables

3.3 Algorithm

We use two passes over the AST during the symbol table generation phase. The first pass generates the top level symbol table and the scoped symbol tables. The second pass checks if the symbol table has been correctly generated. In other words it checks if the symbol table contains the entries for all the identifiers in the current scope and in the scope of its parents.

3.3.1 Symbol Table: Pass One

The first pass is mainly for the generation of the symbol table. No form of checking is done in this pass. All the checking is moved to pass two.

The data-structures manipulated in this pass are the following:

private SymbolTable localsym;

This variable stores the local symbol table at each scope.

private Hashtable astToScopeMap;

This variable stores all the scopes and the corresponding symbol tables

private Hashtable htmlSectionsList;

This variable holds the symbol table created in the scope of html code along with the name of the html body variable.

private Hashtable schemaSectionsList;

This variable holds the symbol table created in the scope of schema

private boolean serviceLevel = true;

This flag indicates whether the scope is in the service block or not.

private boolean sessionLevel = false;

This flag indicates whether the scope is in the session block.

The flags serviceLevel and sessionLevel are used to categorize variables as being serivcevar, sessionvar, or functionvar.

We now look at how these datastructures are built during the first pass of the Symbol table phase.

1. An AnalysisManager object called proxy is passed to the constructor for SymbolTableWalker (First Pass)

2. The object proxy stores the symbol tables created in this phase. It is named so since it is used to provide the datastructres to the other phases of compilation like type checking and code generation.

3. Two functions from the walker class are used to create temporary local symbol tables to store in a hash table of symbol tables and to unscope the local symbol table. Unscoping makes the local symbol table point to its parent indicating that the walker has left a certain scope.

4. Methods for those nodes of the AST that contain elements that need to be inserted into the symbol is written. Typically functions starting with inA... generate a new symbol table, add the symbol name (which is nothing but the name of the identifier), the symbol kind, and the node from the AST.

5. When we are in a new scope level. For instance we enter the service level. We invoke astToScopeMap.put(node,localsym); to insert the local symbol table into astToScopeMap which is a hashtable of symbol tables.

6. Methods starting with an outA... are also defined that unscopes the current symbol table. This makes the local symbol table point to the symbol table in the parent scope.

At the end of pass one of symbol table generation we have all the symbol tables in the astToScopeMap datastructure. We also have some special hashtables like htmlSectionsList and schemaSectionsList which are created for convenience for future passes.

3.3.2 Symbol Table: Pass Two

The second pass during the symbol table phase has a twofold purpose. First, we use it to check if the symbol table has been correctly generated. For this:

1) We visit every node / scope that has affected the creation of the symbol table during pass one.

2) Then, we check if the symbol table contains a definition at the current level or in the parent hierarchy. The compiler generates an error if the symbol is not in the symbol table.

3) Qualified lvalues in tuples are checked in a walker function. inAQualifiedLvalue. We do not immediately know the qualified variable after the dot operator. Therefore depending on where the tuple appears in the program,(i.e. either as a variable assignment or as a function argument) we obtain the type of the variable/argument. We then look up the symbol table that contains the type of the arugment or variable from the schemaSectionsList hashtable. This symbol table is then searched to find the symbol for fieldname that is on the right side of the assignment for the qualified lvalue. A similar check is done when in inAFieldValue walker function. The following code implements the above explanation.

```
public void inAQualifiedLvalue(AQualifiedLvalue node) {
 String symbolName = node.getLeft().getText();
 Symbol symbolLeft = checkIdentifier(symbolName);
 if (symbolLeft == null) {
  errorManager.addSymbolError(node,
    "Tuple variable (left part of qualified lvalue) \""
      + symbolName + "\" is not defined!");
  return;
 } else {
  symAnnotations.put(node.getLeft(), symbolLeft);
 }

 Node tupleVar = symbolLeft.value();
 String fieldName = node.getRight().getText();
 SymbolTable schemasym = null;
 if (tupleVar instanceof AVariable) {
  AVariable tmpVar = (AVariable) tupleVar;
  schemasym = (SymbolTable) schemaSectionsList.get(tmpVar.getType()
    .toString().trim());
 } else if (tupleVar instanceof AArgument) {
  AArgument tmpVar = (AArgument) tupleVar;
  schemasym = (SymbolTable) schemaSectionsList.get(tmpVar.getType()
    .toString().trim());
 }
 Symbol symbolRight = null;
 if (schemasym != null) {
  symbolRight = schemasym.getSymbol(fieldName);
 }
 if (symbolRight == null) {
  errorManager.addSymbolError(node,
    "Field (right part of qualified lvalue) \"" + fieldName
      + "\" is not defined!");
  return;
 } else {
  symAnnotations.put(node.getRight(), symbolRight);
 }
}
```

Second, we create a hashtable called symAnnotations. The hashtable symAnnotations is used during the type checking phase.

3.4 Testing

We developed a pretty printer for symbol tables to verify if all the symbols were correctly inserted and at the correct scope. This was tested for all the benchmarks of 2005.

4 Type Checking

Type checking is the primary task that is carried out during the semantic analysis phase of the compiler. We only perform compile time type checking in other words static type checking.

4.1 Types

We support the following types for our language: int, bool (includes true and false), void, string, true, false, and tuple.

4.2 Type Rules

First we describe the environment in which we specify the typing rules.

Sr - Service (global variables, functions, html and sessions)

Sef - Current session or function

V - Variables in the current session or function

Other notation:

E - Expression S - Statement(s) τ, σ - arbitrary types

4.2.1 Return

The type checking rule for return applies to void functions. If the function is defined as void then the return statement must return a void. In WIG it returns nothing. The return statement should be contained in the body of the function. Hence, we generate an error if the return statement is observed outside the function body.We also generate an error if the return statement is observed for a non-void function.

Type rule:

$\frac{type(Sr,Sef,V)=void}{Se,Sef,V\vdash return}$

Code:

```
public void outAReturnStm(AReturnStm node) {
 if (curFunctionNode == null) {
  errorManager
    .addTypingError(node,
      "in RETURN statement : the return statement is not in a function body");
 } else if (!(curFunctionNode.getType() instanceof AVoidType)) {
  errorManager
    .addTypingError(node,
      "in RETURN statement : must return some value in a non-void function");
 }
}
```

4.2.2 Return Exp

The return expression type checking rule is similar to return. We check if the expression after the return statement has the same the type as the return type specified in the function definition. First, we check if the return statement is inside a function body. Second we check if the return type for the function is a void. In case its a void we produce an error message recommending the programmer to only use return statement. Finally we check if the expression has the same type as the type defined in the function.

Type rule:

$$\frac{Sr,Sef,V \vdash E:\tau \ type(Se,Sef,V)=\sigma \ \sigma:=\tau}{Sr,Sef,V \vdash return \ E}$$

Code:

```
public void outAReturnexpStm(AReturnexpStm node) {
  if (curFunctionNode == null) {
   errorManager
     .addTypingError(node,
       "in RETURN-EXP statement : the return statement is not in a function body");
  } else if (curFunctionNode.getType() instanceof AVoidType) {
   errorManager
     .addTypingError(node,
       "in RETURN-EXP statement : try to return some value in a void function");
  } else {
   PExp returnExp = node.getExp();
   PType returnValueType = nodeType(returnExp);
   if (!assignType(curFunctionNode.getType(), returnValueType)) {
    errorManager
      .addTypingError(
        node,
        "in RETURN-EXP statement : the return type of a function is not compatible with the type of the return value");
   }
   typeTree.put(curFunctionNode, returnValueType);
  }
 }
```

4.2.3 If Statement

When the expression in the condition of an if-statement is evaluated it should always return a boolean value indicating whether the condition was satisfied or not. Any other value the if statement evaluates to (other than true or false) generates an error.

Type Rule:

$$\frac{Sr,Sef,V \vdash E:bool \ Sr,Sef,V \vdash S}{Sr,Sef,V \vdash if(E)S}$$

Code:

```
public void outAIfStm(AIfStm node) {
  if (!isBoolType(nodeType(node.getExp()))) {
   errorManager.addTypingError(node,
     "in IF condition : the type must be BOOL");
  }
 }
```

4.2.4 If-Else Statement

The type rules for the ifelse statement is similar to the if statement.

Type Rule:

$$\frac{Sr,Sef,V \vdash E:bool \;\; Sr,Sef,V \vdash S_1 Sr,Sef,V \vdash S_2}{Sr,Sef,V \vdash if(E)\ S_1\ else\ S_2}$$

Code:

```
public void outAIfelseStm(AIfelseStm node) {
  if (!isBoolType(nodeType(node.getExp()))) {
   errorManager.addTypingError(node,
     "in IF-ELSE condition : the type must be BOOL");
  }
 }
```

4.2.5 While Statement

The expression in the while statement condition should always return a bool.

Type Rule:

$$\frac{Sr,Sef,V \vdash E:bool \;\; Sr,Sef,V \vdash S}{Sr,Sef,V \vdash while(E)S}$$

Code:

```
public void outAWhileStm(AWhileStm node) {
  if (!isBoolType((PType) typeTree.get(node.getExp()))) {
   errorManager.addTypingError(node,
     "in WHILE condition : the type must be BOOL");
  }
 }
```

4.2.6 Boolean binary operators

For the AND and the OR expressions the left and the right sides should be a bool type and the result should also be a bool.

Type Rule (AND):

$$\frac{Sr,Sef,V \vdash E_1:bool \;\; Sr,Sef,V \vdash E_2:bool}{Sr,Sef,V \vdash E_1 \&\& E_2 : bool}$$

Code:

```
 public void outAAndExp(AAndExp node) {
  PType leftType = nodeType(node.getLeft());
  PType rightType = nodeType(node.getRight());
  if (leftType == null || rightType == null) {
   errorManager
     .addTypingError(
       node,
```

```
      "in \"&&\" operation : no type information for at least one side of the operation");
  return;
 }
 if (!isBoolType(leftType)) {
  errorManager.addTypingError(node,
    "in \"&&\" operation : the type must be BOOL");
 }
 if (!isBoolType(rightType)) {
  errorManager.addTypingError(node,
    "in \"&&\" operation : the type must be BOOL");
 }
 typeTree.put(node, boolType);
}
```

Type Rule (OR):

$$\frac{Sr,Sef,V \vdash E_1{:}bool \quad Sr,Sef,V \vdash E_2{:}bool}{Sr,Sef,V \vdash E_1||E_2 : bool}$$

Code:

```
public void outAOrExp(AOrExp node) {
 PType leftType = nodeType(node.getLeft());
 PType rightType = nodeType(node.getRight());
 if (leftType == null || rightType == null) {
  errorManager
    .addTypingError(
      node,
      "in \"||\" operation : no type information for at least one side of the operation");
  return;
 }
 if (!isBoolType(leftType)) {
  errorManager.addTypingError(node,
    "in \"||\" operation : the type must be BOOL");
 }
 if (!isBoolType(rightType)) {
  errorManager.addTypingError(node,
    "in \"||\" operation : the type must be BOOL");
 }
 typeTree.put(node, boolType);
}
```

Type Rule (NOT):

$$\frac{Sr,Sef,V \vdash E_1{:}bool}{Sr,Sef,V \vdash !E_1 : bool}$$

Code:

```
public void outANotExp(ANotExp node) {
 PType leftType = nodeType(node.getLeft());
 if (leftType == null) {
```

```
    errorManager
      .addTypingError(
        node,
        "in \"!\" operation : no type information for at least one side of the operation");
    return;
  }

  if (!isBoolType(leftType)) {
   errorManager.addTypingError(node,
     "in \"!\" operation : the type must be BOOL");
  }
  typeTree.put(node, boolType);
 }
```

4.2.7 Boolean Relational Operators

The relational operators are '==', '! =', '<', '>', '<=', and '>='. For example if a op b indicates that we are applying the relational operator op to expressions a and b. Now, if a and b do not have types we generate and error. For == and ! = the class of a and b have to be identical. For operators other than == and ! = , we report an error if both expressions a and b do not evaluate to integers. The result of the operation is always of type bool.

Type Rule (==):

$$\frac{Sr,Sef,V \vdash E_1{:}int \;\; Sr,Sef,V \vdash E_2{:}int}{Sr,Sef,V\vdash E_1{==}E_2 : bool}$$

Type Rule (! =):

$$\frac{Sr,Sef,V \vdash E_1{:}int \;\; Sr,Sef,V \vdash E_2{:}int}{Sr,Sef,V\vdash E_1{!=}E_2 : bool}$$

Code:

The code for type checking a == and ! = are similar.

```
public void outAXExp(AEqExp node) {
# Here X in the above expression can replaced by: Eq, Neq

  PType leftType = nodeType(node.getLeft());
  PType rightType = nodeType(node.getRight());
  if (leftType == null || rightType == null) {
   errorManager
     .addTypingError(
       node,
       "in \"==\" operation : no type information for at least one side of the operation");
   return;
  }
  if (leftType.getClass() != rightType.getClass()) {
   errorManager
     .addTypingError(node,
       "in \"==\" operation : the type of LHS doesn't match with the type of RHS");
  }
  typeTree.put(node, boolType);
 }
```

Type Rule (<):

$$\frac{Sr,Sef,V \vdash E_1{:}int \;\; Sr,Sef,V \vdash E_2{:}int}{Sr,Sef,V \vdash E_1 < E_2 : bool}$$

Type Rule (>):

$$\frac{Sr,Sef,V \vdash E_1{:}int \;\; Sr,Sef,V \vdash E_2{:}int}{Sr,Sef,V \vdash E_1 > E_2 : bool}$$

Type Rule (<=):

$$\frac{Sr,Sef,V \vdash E_1{:}int \;\; Sr,Sef,V \vdash E_2{:}int}{Sr,Sef,V \vdash E_1 <= E_2 : bool}$$

Type Rule (>=):

$$\frac{Sr,Sef,V \vdash E_1{:}int \;\; Sr,Sef,V \vdash E_2{:}int}{Sr,Sef,V \vdash E_1 >= E_2 : bool}$$

Code:

The code for type checking all the above relational expressions (except $==$ and $!=$ is identical.

```
public void outAXExp(ALteqExp node)
# Here X in the above expression can replaced by: Lteq, Gt, Lt, Gteq

{
 PType leftType = nodeType(node.getLeft());
  PType rightType = nodeType(node.getRight());
  if (leftType == null || rightType == null) {
   errorManager
     .addTypingError(
       node,
       "in \"<\" operation : no type information for at least one side of the operation");
   return;
  }
  if (!isIntType(leftType)) {
   errorManager.addTypingError(node,
     "in \"<\" operation : the type must be INT");
  }
  if (!isIntType(rightType)) {
   errorManager.addTypingError(node,
     "in \"<\" operation : the type must be INT");
  }
  typeTree.put(node, boolType);
 }
```

4.2.8 A Plus Expression

In a plus expression a + b, for instance, we verify 4 different things. First we check if the there is a valid type for the a and b. Second if either one of a or b is a tuple we generate an error message since we cannot apply the + operator to tuples. Third we check if a is an integer/string (since the result has to be an integer/string) and then we check if a and b are compatible using the assignType function.

Type Rule:

$$\frac{Sr,Sef,V \vdash E_1{:}a \;\; Sr,Sef,V \vdash E_2{:}b \;\; a{:=}b}{Sr,Sef,V \vdash E_1 + E_2 : a}$$

Code:

```
public void outAPlusExp(APlusExp node) {
  PType leftType = nodeType(node.getLeft());
  PType rightType = nodeType(node.getRight());
  if (leftType == null || rightType == null) {
   errorManager
     .addTypingError(
       node,
       "in \"+\" operation : no type information for at least one side of the operation");
   return;
  }
  if (isTupleType(leftType) || isTupleType(rightType)) {
   errorManager
     .addTypingError(node,
       "in \"+\" operation : no \"+\" operation for tuple variable");
   return;
  }
  if (isIntType(leftType) && assignType(leftType, rightType)) {
   typeTree.put(node, intType);
   return;
  }
  if (!isStringType(leftType) && !assignType(leftType, rightType)) {
   errorManager
     .addTypingError(node,
       "in \"+\" operation : the type of LHS doesn't match with the type of RHS");
  }

  typeTree.put(node, stringType);
 }
```

4.2.9 Minus, Mult, Div, Mod Expressions

In minus, mult, div and mod expressions of the form a op b, for instance, we verify 2 different things. First we check if the there is a valid type for the a and b. Second we check if a and b are integers since the operation is only valid for integers.

Type Rule($-$):

$$\frac{Sr,Sef,V \vdash E_1{:}int \;\; Sr,Sef,V \vdash E_2{:}int}{Sr,Sef,V \vdash E_1 - E_2 : int}$$

Type Rule($*$):

$$\frac{Sr,Sef,V \vdash E_1{:}int \;\; Sr,Sef,V \vdash E_2{:}int}{Sr,Sef,V \vdash E_1 * E_2 : int}$$

Type Rule($/$):

$$\frac{Sr,Sef,V \vdash E_1{:}int \;\; Sr,Sef,V \vdash E_2{:}int}{Sr,Sef,V \vdash E_1 / E_2 : int}$$

Type Rule($\%$):

$$\frac{Sr,Sef,V \vdash E_1{:}int \;\; Sr,Sef,V \vdash E_2{:}int}{Sr,Sef,V \vdash E_1 \% E_2 : int}$$

Code:

```
public void outAXExp(AModExp node) {
#Here X can be minus, mult, div, mod
 PType leftType = nodeType(node.getLeft());
 PType rightType = nodeType(node.getRight());
 if (leftType == null || rightType == null) {
  errorManager
    .addTypingError(
      node,
      "in \"-\" operation : no type information for at least one side of the operation");
  return;
 }
 if (!isIntType(leftType)) {
  errorManager.addTypingError(node,
    "in \"-\" operation : the type must be INT");
 }
 if (!isIntType(rightType)) {
  errorManager.addTypingError(node,
    "in \"-\" operation : the type must be INT");
 }
 typeTree.put(node, intType);
}
```

4.2.10 Neg Expression

We check if the expression passed to the negation operator is first of all not null and then we check if it is of type integer. If both conditions are not satisfied together then we generate an error message.

Type Rule:

$$\frac{Sr,Sef,V \vdash E:int}{Sr,Sef,V \vdash -E:\ int}$$

Code:

```
public void outANegExp(ANegExp node) {
  PType leftType = nodeType(node.getLeft());
  if (leftType == null) {
   errorManager
     .addTypingError(
       node,
       "in negation operation : no type information for at least one side of the operation");
   return;
  }
  if (!isIntType(leftType)) {
   errorManager.addTypingError(node,
     "in negation operation : the type must be INT");
  }
  typeTree.put(node, intType);
 }
```

4.2.11 Assignment Expression

The type checker checks if in an assignment expression like a=b has comptabile types with respect to expressions a and b. For the basic datatypes we first check if the the assignments are compatible using the following function:

```
 private boolean assignType(PType t1, PType t2) {
  if (t1 == null || t2 == null) {
   return false;
  } else {
   if (t1.getClass() == t2.getClass()) {
    return true;
   } else if (isStringType(t1) && (isIntType(t2) || isBoolType(t2))) {
    return true;
   } else if (isIntType(t1) && isBoolType(t2)) {
    return true;
   } else {
    return false;
   }
  }
 }
```

The compatibilities are checked for the following datatypes.
int =int exp
bool=bool exp
string=string exp
string= int exp
int = bool exp
tuple of schema a = tuple of schema a

Type Rule:

$$\frac{Sr,Sef,V \vdash E_1{:}a \;\; Sr,Sef,V \vdash E_2{:}b}{Sr,Sef,V \vdash E_1{=}E_2{:}a}$$

Checking assignment for tuples is a bit more complicated. For tuples we do not immediately know the datatype of the fields.
1. First we check if both the left hand side and the right hand side objects are of tuple type.
2. If the identifier names are different and the RHS is of intermediate type:
— 2.1 We obtain the symbol table for the schema for the tuple on the LHS
— 2.2 An error is reported if there exists no schema symbol table for the LHS tuple.
3. Else:
— 3.1 We map the RHS tuple to an intermediate tuple type.
— 3.2 We obtain the symbol table for the intermediate type and run a check with respect to the LHS tuple. First we check if the number of fields are the same. Then an iterator goes through the fields and the types for each of these fields are checked.

Code: For code please refer to TypeCheckingWalker.java in the package (reason: Space limitation).

Code for creation of intermediate type:

```
public void outATupleExp(ATupleExp node) {
  PType nodeType = variableType(((AAssignExp) node.parent()).getLvalue());
  Hashtable fieldsMap = new Hashtable();
  List fieldvalueList = (List) node.getFieldvalue();
  Iterator fieldvalueIter = fieldvalueList.iterator();
```

```
 while (fieldvalueIter.hasNext()) {
  AFieldvalue fieldvalue = (AFieldvalue) fieldvalueIter.next();
  PType fieldType = nodeType(fieldvalue.getExp());
  PType interType;
  if (fieldType instanceof ABoolType) {
   interType = new ABoolType(new TBool());
  } else if (fieldType instanceof AIntType) {
   interType = new AIntType(new TInt());
  } else if (fieldType instanceof AStringType) {
   interType = new AStringType(new TString());
  } else {
   interType = null;
  }
  AField interField = new AField(interType, new TIdentifier(
    fieldvalue.getIdentifier().getText()));
  fieldsMap.put(fieldvalue.getIdentifier().getText(), interField);
 }

 AIntermidiateTupleType interTupleType = new AIntermidiateTupleType(
   fieldsMap);
 interTypeTree.put(node, interTupleType);

 ATupleType resultType = new ATupleType(new TIdentifier(
   "__intermidiate_type"));

 if (nodeType != null) {
  typeTree.put(node, resultType);
 }
}
```

4.2.12 Call Expression

The number of function parameters must be the same. They must have the same order and type as defined in the funciton defintion.

Type Rule:

$$\frac{Sr,Sef,V \vdash E{:}a \;\; Sr,Sef,V \vdash E_i{:}p_i \;\; type(Sr,F)=a \;\; argtype(Se,F,i)=y_i \;\; y_i=p_i}{Sr,Sef,V \vdash E.F(E_1,E_2,\ldots,E_n){:}a}$$

Code:

```
public void outACallExp(ACallExp node) {
 Symbol s = (Symbol) curSymbolTable.getSymbol(node.getIdentifier()
   .getText());
 if (s == null) {
  errorManager.addTypingError(node,
    "in function call : a function name is required");
  return;
 }
 List argList = ((AFunction) s.value()).getArgument();
 List paramList = node.getExp();
 if (argList.size() != paramList.size()) {
  errorManager
```

```
    .addTypingError(
      node,
      "in function call : the number of arguments doesn’t match with the number of parameters");
  return;
 }
 for (int i = 0; i < argList.size(); i++) {
  if (((AArgument) argList.get(i)).getType().getClass() != ((PType) typeTree
    .get(paramList.get(i))).getClass()) {
   errorManager
     .addTypingError(
       node,
       "in function call : the type of argument doesn’t match with the type of parameter");
   return;
  }
 }
 typeTree.put(node, ((AFunction) s.value()).getType());
}
```

4.2.13 Field Value

field value of tuple = field value of tuple
We check if the basic type of the field values of a tuples match. We do not support tuple objects inside tuples.

Type Rule:

$$\frac{Sr,Sef,V \vdash E_1.x{:}a \;\; Sr,Sef,V \vdash E_2.x{:}b}{Sr,Sef,V \vdash E_1.x{=}E_2.x{:}a}$$

Code:

```
public void outAFieldvalue(AFieldvalue node) {
 PType lvalueType = variableType(node);
 PType rightType = nodeType(node.getExp());
 if (lvalueType == null || rightType == null) {
  if (rightType == null) {
   errorManager
     .addTypingError(node,
       "in field assignment : no type information for RHS of the operation");
  }
  if (lvalueType == null) {
   errorManager
     .addTypingError(node,
       "in field assignment : no type information for LHS of the operation");
  }
  return;
 }
 if (!assignType(lvalueType, rightType)) {
  errorManager
    .addTypingError(node,
      "in field assignment : types of assignment are not compatible");
 }
 typeTree.put(node, lvalueType);
}
```

4.2.14 Tuple keep operation

The keep operation on a tuple creates a new tuple that extracts one identifier from the tuple. The keep operation creates a new tuple typle with only one field. This operation will generate an intermediate tuple type.

Type Rule:

$$\frac{Sr,Sef,V \vdash T1{:}t1 \;\; Sr,Sef,V \vdash A{:}a \;\; type(T.A){:=}type(A)}{Sr,Sef,V \vdash E_1{=}T1\backslash{+}A{:}T}$$

Code:

```
public void outAKeepExp(AKeepExp node) {
  PType leftType = nodeType(node.getLeft());
  if (leftType == null) {
   errorManager
     .addTypingError(node,
       "in \"\\+\" operation : no type information for LHS of the operation");
   return;
  }
  if (!isTupleType(leftType)) {
   errorManager.addTypingError(node,
     "in \"\\+\" operation : the type must be TUPLE");
  } else {
   Map fieldsMap = new Hashtable();

   String rightFieldName = node.getIdentifier().getText();
   ATupleType leftTupleType = (ATupleType) leftType;
   if (leftTupleType.getIdentifier().getText() != "__intermidiate_type") {

    SymbolTable schemasym = getSymbolTableOfSchema(((ALvalueExp) node
      .getLeft()).getLvalue());
    if (schemasym == null) {
     errorManager
       .addTypingError(node,
         "in \"\\+\" operation : no symbol information for the LHS schema");
    } else {
     Symbol rightFieldSymbol = schemasym
       .getSymbol(rightFieldName);
     if (rightFieldSymbol == null) {
      errorManager
        .addTypingError(
          node,
          "in \"\\+\" operation : try to keep a field which doesn't exist in the source schema");
     } else {
      AField fieldNode = (AField) rightFieldSymbol.value();
      fieldsMap.put(fieldNode.getIdentifier().getText(),
        fieldNode);
     }
    }
   } else {
    AIntermidiateTupleType interTupleType = (AIntermidiateTupleType) interTypeTree
      .get(node.getLeft());
    Map interFieldMap = interTupleType.getFieldsMap();
```

```
    AField interFieldNode = (AField) interFieldMap
      .get(rightFieldName);
    if (interFieldNode == null) {
     errorManager
       .addTypingError(
         node,
         "in \"\\+\" operation : try to keep a field which doesn’t exist in the source schema");
    } else {
     fieldsMap.put(interFieldNode.getIdentifier().getText(),
       interFieldNode);
    }
   }

   AIntermidiateTupleType interTupleType = new AIntermidiateTupleType(
     fieldsMap);
   interTypeTree.put(node, interTupleType);
  }

  ATupleType resultType = new ATupleType(new TIdentifier(
    "__intermidiate_type"));

  typeTree.put(node, resultType);
 }
```

4.2.15 Tuple remove operation

The remove operation on a tuple creates a new tuple that extracts all but one identifier from the tuple. The remove operation creates a new tuple typle with n-1 fields. Here n is the total number of fields. This operation will generate an intermediate tuple type.

Type Rule:

$$\frac{Sr,Sef,V \vdash T1.i{:}t1_i \;\; Sr,Sef,V \vdash A{:}a \;\; Sr,Sef,V \vdash T_i{:}t1_i,i{\neq}A}{Sr,Sef,V{\vdash}\; E_1{=}T1\backslash{-}A{:}T}$$

Code:

```
public void outARemoveExp(ARemoveExp node) {
  PType leftType = nodeType(node.getLeft());
  if (leftType == null) {
   errorManager
     .addTypingError(node,
       "in \"\\-\" : no type information for LHS of the operation");
   return;
  }
  if (!isTupleType(leftType)) {
   errorManager.addTypingError(node,
     "in \"\\-\" operation : the type must be TUPLE");
  } else {
   Map fieldsMap = new Hashtable();

   String rightFieldName = node.getIdentifier().getText();
   ATupleType leftTupleType = (ATupleType) leftType;
   if (leftTupleType.getIdentifier().getText() != "__intermidiate_type") {
```

```
 SymbolTable schemasym = getSymbolTableOfSchema(((ALvalueExp) node
   .getLeft()).getLvalue());
 if (schemasym == null) {
  errorManager
    .addTypingError(node,
      "in \"\\-\" operation : no symbol information for the LHS schema");
 } else {
  String fieldName = node.getIdentifier().getText();
  Symbol fieldSymbol = schemasym.getSymbol(fieldName);
  if (fieldSymbol == null) {
   errorManager
     .addTypingError(
       node,
       "in \"\\-\" operation : try to remove a field which doesn't exist in the source schema");
  } else {
   Enumeration fields = schemasym.elements();
   while (fields.hasMoreElements()) {
    Symbol element = (Symbol) fields.nextElement();
    if (element != fieldSymbol) {
     AField fieldNode = (AField) element.value();
     fieldsMap.put(fieldNode.getIdentifier()
       .getText(), fieldNode);
    }
   }
  }
 }
} else {
 AIntermidiateTupleType interTupleType = (AIntermidiateTupleType) interTypeTree
   .get(node.getLeft());

 Iterator interFields = interTupleType.getFieldsMap().values()
   .iterator();
 boolean matched = false;
 while (interFields.hasNext()) {
  AField interFieldNode = (AField) interFields.next();
  String interFieldName = interFieldNode.getIdentifier()
    .getText();
  if (interFieldName.equals(rightFieldName)) {
   matched = true;
  } else {
   fieldsMap.put(interFieldNode.getIdentifier().getText(),
     interFieldNode);
  }
 }
 if (!matched) {
  fieldsMap.clear();
  errorManager
    .addTypingError(
      node,
      "in \"\\-\" operation : try to remove a field which doesn't exist in the source schema");
 }
}
```

```
  AIntermidiateTupleType interTupleType = new AIntermidiateTupleType(
    fieldsMap);
  interTypeTree.put(node, interTupleType);
 }

 ATupleType resultType = new ATupleType(new TIdentifier(
   "__intermidiate_type"));

 typeTree.put(node, resultType);
}
```

4.2.16 Tuple keep many operation

The tuple keep many operation is an extension to the tuple keep operation. It allows the WIG programmer to create a new tuple with more than one identifiers extracted from an old tuple.This operation will generate an intermediate tuple type.

Type Rule:

$$\frac{Sr,Sef,V \vdash T1{:}t1 \;\; Sr,Sef,V \vdash A_j{:}a_j \;\; type(T.A_j){:=}type(A_j)}{Sr,Sef,V \vdash E_1{=}T1\backslash{+}(A_1,A_2,\ldots A_k){:}T}$$

Code:

```
public void outAKeepManyExp(AKeepManyExp node) {
 PType leftType = nodeType(node.getLeft());
 if (leftType == null) {
  errorManager
    .addTypingError(node,
      "in \"\\+ (...)\" operation : no type information for LHS of the operation");
  return;
 }
 if (!isTupleType(leftType)) {
  errorManager.addTypingError(node,
    "in \"\\+ (...)\" operation : the type must be TUPLE");
 } else {
  Map fieldsMap = new Hashtable();

  Map leftFieldsMap = new Hashtable();
  ATupleType leftTupleType = (ATupleType) leftType;
  if (leftTupleType.getIdentifier().getText() != "__intermidiate_type") {
   SymbolTable leftSchemasym = getSymbolTableOfSchema(((ALvalueExp) node
     .getLeft()).getLvalue());
   if (leftSchemasym == null) {
    errorManager
      .addTypingError(node,
        "in \"\\+ (...)\" operation : no symbol information for LHS of the operation");
   } else {
    Enumeration fieldsLeft = leftSchemasym.elements();
    while (fieldsLeft.hasMoreElements()) {
     Symbol fieldSymbol = (Symbol) fieldsLeft.nextElement();
     AField fieldNode = (AField) fieldSymbol.value();
     leftFieldsMap.put(fieldNode.getIdentifier().getText(),
       fieldNode);
    }
```

```
   }
  } else {
   AIntermidiateTupleType leftInterTupleType = (AIntermidiateTupleType) interTypeTree
     .get(node.getLeft());
   if (leftInterTupleType == null) {
    errorManager
      .addTypingError(node,
        "in \"\\+ (...)\" operation : no symbol information for LHS of the operation");
   } else {
    leftFieldsMap = leftInterTupleType.getFieldsMap();
   }
  }

  List rightFields = node.getIdentifier();
  if (leftFieldsMap.size() < rightFields.size()) {
   errorManager
     .addTypingError(node,
       "in \"\\+ (...)\" operation : the LHS schema has less fields than RHS");
  } else {
   Iterator rightFieldsIter = rightFields.iterator();
   while (rightFieldsIter.hasNext()) {
    TIdentifier rifhtFieldId = (TIdentifier) rightFieldsIter
      .next();
    String rightFieldName = rifhtFieldId.getText();

    AField interFieldNode = (AField) leftFieldsMap
      .get(rightFieldName);
    if (interFieldNode == null) {
     fieldsMap.clear();
     errorManager
       .addTypingError(
         node,
         "in \"\\+ (...)\" operation : try to keep a field which doesn't exist in the source schema");
    } else {
     fieldsMap.put(interFieldNode.getIdentifier().getText(),
       interFieldNode);
    }
   }
  }

  AIntermidiateTupleType interTupleType = new AIntermidiateTupleType(
    fieldsMap);
  interTypeTree.put(node, interTupleType);
 }

 ATupleType resultType = new ATupleType(new TIdentifier(
   "__intermidiate_type"));

 typeTree.put(node, resultType);
}
```

4.2.17 Tuple remove many operation

The remove many operation on a tuple creates a new tuple that extracts all but n-k (k¡n) identifiers from the source tuple. The remove operation creates a new tuple typle with n-k fields. Here n is the total number of fields and k is the number of fields removed. This operation will generate an intermediate tuple type.

Type Rule:

$$\frac{Sr,Sef,V \vdash T1.i{:}t1_i \;\; Sr,Sef,V \vdash A_j{:}a_j \;\; Sr,Sef,V \vdash T_i{:}t1_i,i{\neq}(A_j)}{Sr,Sef,V{\vdash}\; E_1{=}T1\backslash{-}(A_1,A_2,...A_k){:}T}$$

Code:

```
public void outARemoveManyExp(ARemoveManyExp node) {
  PType leftType = nodeType(node.getLeft());
  if (leftType == null) {
   errorManager
     .addTypingError(node,
       "in \"\\- (...)\" operation : no type information for LHS of the operation");
   return;
  }
  if (!isTupleType(leftType)) {
   errorManager
     .addTypingError(node,
       "in \"\\- (...)\" operation : the type of LHS must be TUPLE");
  } else {
   Map fieldsMap = new Hashtable();

   Map leftFieldsMap = new Hashtable();
   ATupleType leftTupleType = (ATupleType) leftType;
   if (leftTupleType.getIdentifier().getText() != "__intermidiate_type") {
    SymbolTable leftSchemasym = getSymbolTableOfSchema(((ALvalueExp) node
      .getLeft()).getLvalue());
    if (leftSchemasym == null) {
     errorManager
       .addTypingError(node,
         "in \"\\- (...)\" operation : no symbol information for LHS of the operation");
    } else {
     Enumeration fieldsLeft = leftSchemasym.elements();
     while (fieldsLeft.hasMoreElements()) {
      Symbol fieldSymbol = (Symbol) fieldsLeft.nextElement();
      AField fieldNode = (AField) fieldSymbol.value();
      leftFieldsMap.put(fieldNode.getIdentifier().getText(),
        fieldNode);
     }
    }
   } else {
    AIntermidiateTupleType leftInterTupleType = (AIntermidiateTupleType) interTypeTree
      .get(node.getLeft());
    if (leftInterTupleType == null) {
     errorManager
       .addTypingError(node,
         "in \"\\- (...)\" operation : no symbol information for LHS of the operation");
    } else {
     leftFieldsMap = leftInterTupleType.getFieldsMap();
```

```
    }
   }

   List rightFields = node.getIdentifier();
   if (leftFieldsMap.size() < rightFields.size()) {
    errorManager
      .addTypingError(node,
        "in \"\\- (...)\" operation : the LHS schema has less fields than RHS");
   } else {
    fieldsMap.putAll(leftFieldsMap);

    Iterator rightFieldsIter = rightFields.iterator();
    while (rightFieldsIter.hasNext()) {
     TIdentifier rifhtFieldId = (TIdentifier) rightFieldsIter
       .next();
     String rightFieldName = rifhtFieldId.getText();

     AField interFieldNode = (AField) leftFieldsMap
       .get(rightFieldName);
     if (interFieldNode == null) {
      fieldsMap.clear();
      errorManager
        .addTypingError(
          node,
          "in \"\\- (...)\" operation : try to remove a field which doesn't exist in the source schema");
     } else {
      fieldsMap.remove(rightFieldName);
     }
    }
   }

   AIntermidiateTupleType interTupleType = new AIntermidiateTupleType(
     fieldsMap);
   interTypeTree.put(node, interTupleType);
  }

  ATupleType resultType = new ATupleType(new TIdentifier(
    "__intermidiate_type"));

  typeTree.put(node, resultType);
 }
```

4.2.18 Tuple join operation

During the join operation t3=t1¡¡t2 between two tuples t1 and t2 some conditions need to be met with. t1 and t2 must agree with the attributes they have in common. The resultant tuple t3 must have a union of fields from t1 and t2. When two fields in t1 and t2 are the same we check if they have the same basic type. If both the fields are the same we choose the field from t2. In the following rule n is the total number of attributes. This operation will generate an intermediate tuple type.

Type Rule:

$$\frac{Sr,Sef,V \vdash T1.i{:}t1_i \;\; Sr,Sef,V \vdash T2.j{:}t2_j \;\; Sr,Sef,V \vdash T3_i{:}t1_i \;\; T3_j{:}t2_j,\ i\ is\ from\ T1\ and\ j\ is\ from\ T2}{Sr,Sef,V \vdash T3{=}T1{<<}T2{:}type(T3)}$$

Code:

```
public void outAJoinExp(AJoinExp node) {
  PType leftType = nodeType(node.getLeft());
  PType rightType = nodeType(node.getRight());
  if (leftType == null || rightType == null) {
   errorManager
     .addTypingError(
       node,
       "in \"<<\" operation : no type information for at least one side of the operation");
   return;
  }
  if (!isTupleType(leftType) || !isTupleType(rightType)) {
   errorManager
     .addTypingError(node,
       "in \"<<\" operation : the type of either side must be TUPLE");
  } else {
   Map fieldsMap = new Hashtable();

   Map leftFieldsMap = new Hashtable();
   Map rightFieldsMap = new Hashtable();

   ATupleType leftTupleType = (ATupleType) leftType;
   ATupleType rightTupleType = (ATupleType) rightType;
   if (leftTupleType.getIdentifier().getText() != "__intermidiate_type") {

    SymbolTable leftSchemasym = getSymbolTableOfSchema(((ALvalueExp) node
      .getLeft()).getLvalue());
    if (leftSchemasym == null) {
     errorManager
       .addTypingError(node,
         "in \"<<\" operation : no symbol information for LHS of the operation");
    } else {
     Enumeration fieldsLeft = leftSchemasym.elements();
     while (fieldsLeft.hasMoreElements()) {
      Symbol fieldSymbol = (Symbol) fieldsLeft.nextElement();
      AField fieldNode = (AField) fieldSymbol.value();
      leftFieldsMap.put(fieldNode.getIdentifier().getText(),
        fieldNode);
     }
    }
   } else {
    AIntermidiateTupleType leftInterTupleType = (AIntermidiateTupleType) interTypeTree
      .get(node.getLeft());
    if (leftInterTupleType == null) {
     errorManager
       .addTypingError(node,
         "in \"<<\" operation : no symbol information for LHS of the operation");
    } else {
     leftFieldsMap = leftInterTupleType.getFieldsMap();
    }
   }

   if (rightTupleType.getIdentifier().getText() != "__intermidiate_type") {
```

```
 SymbolTable rightSchemasym = getSymbolTableOfSchema(((ALvalueExp) node
   .getRight()).getLvalue());
 if (rightSchemasym == null) {
  errorManager
    .addTypingError(node,
      "in \"<<\" operation : no symbol information for RHS of the operation");
 } else {
  Enumeration fieldsRight = rightSchemasym.elements();
  while (fieldsRight.hasMoreElements()) {
   Symbol fieldSymbol = (Symbol) fieldsRight.nextElement();
   AField fieldNode = (AField) fieldSymbol.value();
   rightFieldsMap.put(fieldNode.getIdentifier().getText(),
     fieldNode);
  }
 }
} else {
 AIntermidiateTupleType rightInterTupleType = (AIntermidiateTupleType) interTypeTree
   .get(node.getRight());
 if (rightInterTupleType == null) {
  errorManager
    .addTypingError(node,
      "in \"<<\" operation : no symbol information for RHS of the operation");
 } else {
  rightFieldsMap = rightInterTupleType.getFieldsMap();
 }
}

{
 fieldsMap.putAll(leftFieldsMap);

 Iterator rightFieldsIter = rightFieldsMap.values().iterator();
 while (rightFieldsIter.hasNext()) {
  AField rightFieldNode = (AField) rightFieldsIter.next();

  AField existFieldNode = (AField) fieldsMap
    .get(rightFieldNode.getIdentifier().getText());
  if (existFieldNode != null
    && existFieldNode.getType().getClass() != rightFieldNode
      .getType().getClass()) {
   fieldsMap.clear();
   errorManager.addTypingError(node,
     "in \"<<\" operation : both sides of operation have the same field \""
       + rightFieldNode.getIdentifier()
         .getText()
       + "\", but with different types");

  } else {
   fieldsMap.put(rightFieldNode.getIdentifier().getText(),
     rightFieldNode);
  }
 }

}
```

```
    AIntermidiateTupleType interTupleType = new AIntermidiateTupleType(
      fieldsMap);
    interTypeTree.put(node, interTupleType);
  }

  ATupleType resultType = new ATupleType(new TIdentifier(
    "__intermidiate_type"));

  typeTree.put(node, resultType);
}
```

Apart from the code listed in the report we used several utility functions to facilitate type checking for tuples and other types. Please refer to the type checking walker class available with the code.

4.3 Testing

We tested type checking by introducing type errors in benchmarks provided to us this year. Also, we thoroughly applied type checking to tuples which are a complex type. The tuple torture wig program was run successfully. We also introduced type errors in tuple torture to generate errors.

5 Code Generation

5.1 Strategy

The code generated for the input WIG program is in Python, a very nice script language with full Object-Oriented programming features. The code generation strategy involves one pass over the abstract syntax tree of the WIG program. The code generation walker generates and stores pertinent python code at each node. The final code is created at the service node. This happens when the outAService function is called. In figure 5.1 we describe the overall strategy. The outAservice method in code generation walker first creates the html headers. Then the definitions for the constant html code is added. We then include the function definitions. The schemas are defined after this. The session defintions are then appended to the generated code. To allow execution of the program a main method is included. Finally, the CGI script is written to the file.

Since, service is a toplevel method the depthfirst adapter goes to the leaf nodes first and then goes bottom up. As a result we are able to generate code for parts of the program at the nodes before the service node is reached. For instance, at the html node we create a python function called generateHtml¡identifier¿ to create the html document from the body of the html identifier.

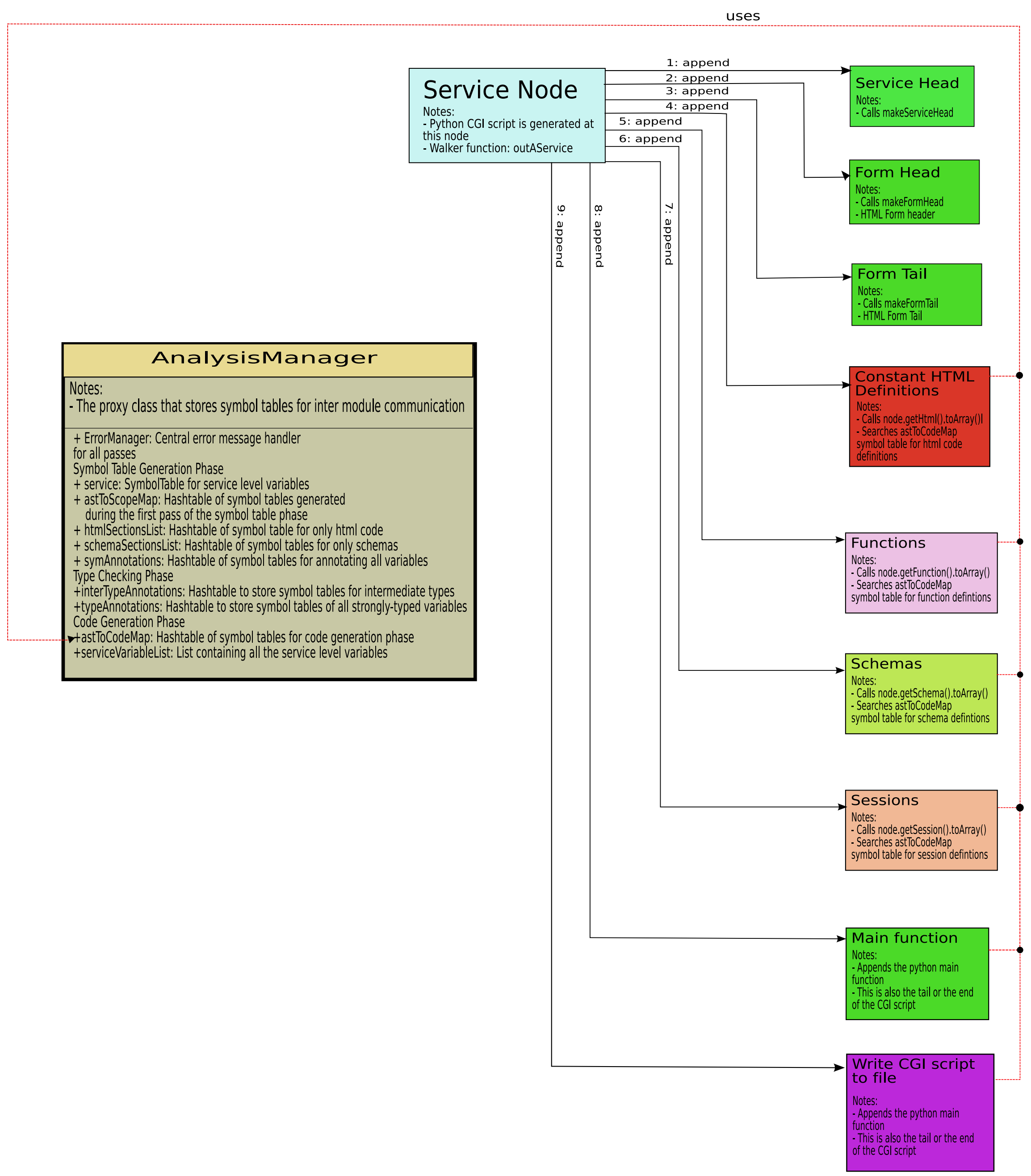

Figure 2: Overall code generation process from the service node

5.2 Code Templates

$WIGElement$	$PythonScript$	$ScopeinScipt$
x	x	x
$Foo.wig$	$classServiceFoo:$	$global$
$consthtmlFoo$	$defgenerateHtmlFoo():$	$ServiceFoo'smethod$
$schemaFoo$	$classSchemaFoo:$	$global$
$tupleuserinfoFoo;$	$self.Foo$	$ServiceFoo'sattribute$
$boolFoo$	$defFoo():$	$ServiceFoo'smethod$
$sessionFoo()$	$classSessionFooFoo:$	$global$
$variableinsession$	$attributeofsessionclass$	$insession$
$variableinfunction$	$localvariableofmethod$	$inmethod$
N/A	$defmain():$	$global$
$intermediatetupletype$	$classInterSchema_UID:$	$global$
N/A	$classInstruction:$	$global$

The python code that is generated from our compiler has the following

WIG program:

```
service {
 const html Card_Info = <html>
<head>
<title>WIG PostCard Generator</title>
</head>
<body>
<br>WIG PostCard Generator<br>
<br>
Authors: Sagar Sen and Ximeng Sun<br>
<br>
Sender Name:
<input name = "sender_name" size="40" type="text"><br>
Receipient Name:
<input name="recepient_name" size="40" type="text"><br>
Greeting:
<input name="greeting" size="40" type="text"><br>
Message Body: <br>
<input name="message" value="" cols=50 rows=50 type="text"><br>
<br>
</body>
</html>;

 const html Card_Gen =<html>
<head>
<title>Greeting Card</title>
</head>
<body>
<table border="1" cellpadding="2"
cellspacing="2">
<tbody>
<tr>
<td><[greeting]><[recepient_name]><br>
<br>
<[message]><br>
<br>
```

```
<br>
<[sender_name]><br>
</td>

<td><img alt="Postcard Pic" src="http://www.cs.mcgill.ca/~ssen3/postcard_pics/lancre.jpg"><br>
</td>
</tr>
</tbody>
</table>
<br>
</body>
</html>;

schema userinfo {
 string sender_name;
 string recepient_name;
 string greeting;
 string message;
}

schema userinfo2 {
 string sender_name;
 string recepient_name;
 string greeting;
 string message;
}

tuple userinfo tmp;

bool foo(tuple userinfo t1, tuple userinfo2 t2){
 return t1 == t2;
}

session GetInfo() {
 string  s1, s2, s3, s4;
 tuple userinfo user;
 tuple userinfo2 user2;
 int pc;

 pc = 0;
 while (pc < 2){
  pc = pc + 1;
  show Card_Info receive[s1=sender_name,s2=recepient_name,s3=greeting,s4=message];
 }
 user = tuple{sender_name=s1, recepient_name=s2, greeting=s3, message=s4};
 user2 = user \+ (sender_name, recepient_name, greeting, message);
 foo(user, user2);
 exit plug Card_Gen[greeting=user2.sender_name,recepient_name=user2.recepient_name,
      sender_name=user2.sender_name, message=user2.message];
}
}
```

Generated Python script:

```
#!/usr/local/bin/python

import time
import string
import random
import os
import cgi
import re
import cgitb;cgitb.enable()
import pickle

class Instruction:
  def __init__(self, stm, type="other", offset=0):
    self.type = type
    self.stm = stm
    self.offset = offset

  def __str__(self):
    rtstr = self.type +":"+ str(self.offset)+":"+ self.stm
    return rtstr

  def __repr__(self):
    rtstr = self.type +":"+ str(self.offset)+":"+ self.stm
    return rtstr

class ServiceBench:
  def __init__(self):
    self.tmp = Schemauserinfo()

  def __getstate__(self):
    odict = self.__dict__.copy()
    del odict['form']
    del odict['url']
    return odict

  def getField(self, field):
    return self.form.getvalue(field)

  def generateRandomSessionid(self, session):
    random.seed(time.time())
    d = [random.choice(string.letters) for x in xrange(20)]
    s = "".join(d)
    return session + "$" + s

  def dumpSession(self, session):
    pickle.dump(session,open(session.sessionid,'wb'))

  def loadSession(self, sessionid):
    session = pickle.load(open(sessionid))
    session.service = self
    session.form = self.form
    session.sessionid = sessionid
    session._condition_flag = False
```

```
    return session

  def run(self):
    self.form = cgi.FieldStorage(keep_blank_values=True)
    self.url = "http://" + str(os.getenv("SERVER_NAME")) + str(os.getenv("SCRIPT_NAME"))
    sessionid = os.getenv("QUERY_STRING")
    if sessionid == "GetInfo":
      session_Bench_GetInfo = SessionBenchGetInfo(self, sessionid, self.form)
      session_Bench_GetInfo.run()
      self.dumpSession(session_Bench_GetInfo)
    if sessionid[:8] == "GetInfo$":
      session_Bench_GetInfo = self.loadSession(sessionid)
      session_Bench_GetInfo.run()
      self.dumpSession(session_Bench_GetInfo)

  def makeFormHead(self, sessionid):
    print "Content-type:text/html"
    print
    print "<form method=\"POST\" action=\"%s?%s\">" % (self.url, sessionid)
    print

  def makeFormTail(self):
    print "<p><input type=\"submit\" value=\"continue\">"
    print "</form>"
    print

  def generateHtmlCard_Info (self):
    # actual code is omitted for describing the template

  def generateHtmlCard_Gen (self, greeting, recepient_name, message, sender_name):
    # actual code is omitted for describing the template

  def foo(self, t1, t2):
    return t1 == t2

class Schemauserinfo:
  def __init__(self):
    self.sender_name = ""
    self.recepient_name = ""
    self.greeting = ""
    self.message = ""

  def setAll(self, sender_name, recepient_name, greeting, message):
    self.sender_name = sender_name
    self.recepient_name = recepient_name
    self.greeting = greeting
    self.message = message

  def __eq__(self, other):
    if (type(self.sender_name) != type(other.sender_name)) or (self.sender_name != other.sender_name):
      return False
    if (type(self.recepient_name) != type(other.recepient_name)) or (self.recepient_name != other.recepient_name):
      return False
```

```
    if (type(self.greeting) != type(other.greeting)) or (self.greeting != other.greeting):
      return False
    if (type(self.message) != type(other.message)) or (self.message != other.message):
      return False
    return True

  def __ne__(self, other):
    if (type(self.sender_name) != type(other.sender_name)) or (self.sender_name != other.sender_name):
      return True
    if (type(self.recepient_name) != type(other.recepient_name)) or (self.recepient_name != other.recepient_name):
      return True
    if (type(self.greeting) != type(other.greeting)) or (self.greeting != other.greeting):
      return True
    if (type(self.message) != type(other.message)) or (self.message != other.message):
      return True
    return False

  def assign(self, other):
    self.sender_name = other.sender_name
    self.recepient_name = other.recepient_name
    self.greeting = other.greeting
    self.message = other.message

class Schemauserinfo2:
    # actual code is omitted for describing the template

class SessionBenchGetInfo:
  def __init__(self, service, sessionid, form):
    self.form = form
    self.service = service
    self.sessionid = sessionid
    self._condition_flag = False

    self.GetInfo_s1 = ""
    self.GetInfo_s2 = ""
    self.GetInfo_s3 = ""
    self.GetInfo_s4 = ""
    self.GetInfo_user = Schemauserinfo()
    self.GetInfo_user2 = Schemauserinfo2()
    self.GetInfo_pc = 0
    self.pc = 0
    self.instructionStack = [
    Instruction("self.sessionid = self.service.generateRandomSessionid('GetInfo')"),
    Instruction("self.GetInfo_pc = 0"),
    Instruction("self._condition_flag = (self.GetInfo_pc < 2)", "while", 8),
    Instruction("self.GetInfo_pc = self.GetInfo_pc + 1"),
    Instruction("self.service.makeFormHead(self.sessionid)"),
    Instruction("self.service.generateHtmlCard_Info()"),
    Instruction("self.service.makeFormTail()"),
    Instruction("return", "return"),
    Instruction("self.getFields([\"self.GetInfo_s1 = self.form.getvalue('sender_name')\", \"self.GetInfo_s2 = self.form.getvalue('recepient_name')\", \"self.GetInfo_s3 = self.form.getvalue('greeting')\", \"self.GetInfo_s4 = self.form.getvalue('message')\"])"),
    Instruction("goto", "goto", -7),
    Instruction("self.GetInfo_user.assign(InterSchema_28910606(self.GetInfo_s1, self.GetInfo_s2, self.GetInfo_s3, self.GetInfo_s4))"),
```

```
    Instruction("self.GetInfo_user2.assign(InterSchema_1187613(self.GetInfo_user.message, self.GetInfo_user.greeting, self.GetInfo_user.recepient_name, self.GetInfo_user.sender_name))"),
    Instruction("self.service.foo(self.GetInfo_user,self.GetInfo_user2)"),
    Instruction("print 'Content-type: text/html'"),
    Instruction("print"),
    Instruction("print"),
    Instruction("self.service.generateHtmlCard_Gen(self.GetInfo_user2.sender_name, self.GetInfo_user2.recepient_name, self.GetInfo_user2.sender_name, self.GetInfo_user2.message)"),
    Instruction("return", "return"),
    ]

  def __getstate__(self):
    odict = self.__dict__.copy()
    del odict['form']
    del odict['service']
    del odict['sessionid']
    del odict['_condition_flag']
    return odict

  def getFields(self, fieldvalue_list):
    for fieldvalue in fieldvalue_list:
      exec fieldvalue

  def getNextInstruction(self):
    if self.pc < len(self.instructionStack):
      return self.instructionStack[self.pc]
    else:
      return None

  def run(self):
    while(True):
      nextInstruction = self.getNextInstruction()
      if nextInstruction == None:
        return

      fh = open('wigvm.log', 'ab')
      fh.write("%d:%s;" % (self.pc, nextInstruction))
      fh.close()

      if (nextInstruction.type == "if") or (nextInstruction.type == "ifelse") or (nextInstruction.type == "while"):
        exec str(nextInstruction.stm)
        if self._condition_flag == True:
          self.pc = self.pc + 1
        else:
          self.pc = self.pc + nextInstruction.offset
      if nextInstruction.type == "other":
        exec str(nextInstruction.stm)
        self.pc = self.pc + 1
      if nextInstruction.type == "return":
        self.pc = self.pc + 1
        return
      if nextInstruction.type == "goto":
        self.pc = self.pc + nextInstruction.offset

class InterSchema_28910606:
```

```
  def __init__(self, message, greeting, recepient_name, sender_name):
    self.message = message
    self.greeting = greeting
    self.recepient_name = recepient_name
    self.sender_name = sender_name

  def setAll(self, message, greeting, recepient_name, sender_name):
    self.message = message
    self.greeting = greeting
    self.recepient_name = recepient_name
    self.sender_name = sender_name

  def __eq__(self, other):
    if (type(self.message) != type(other.message)) or (self.message != other.message):
      return False
    if (type(self.greeting) != type(other.greeting)) or (self.greeting != other.greeting):
      return False
    if (type(self.recepient_name) != type(other.recepient_name)) or (self.recepient_name != other.recepient_name):
      return False
    if (type(self.sender_name) != type(other.sender_name)) or (self.sender_name != other.sender_name):
      return False
    return True

  def __ne__(self, other):
    if (type(self.message) != type(other.message)) or (self.message != other.message):
      return True
    if (type(self.greeting) != type(other.greeting)) or (self.greeting != other.greeting):
      return True
    if (type(self.recepient_name) != type(other.recepient_name)) or (self.recepient_name != other.recepient_name):
      return True
    if (type(self.sender_name) != type(other.sender_name)) or (self.sender_name != other.sender_name):
      return True
    return False

  def assign(self, other):
    self.message = other.message
    self.greeting = other.greeting
    self.recepient_name = other.recepient_name
    self.sender_name = other.sender_name

class InterSchema_1187613:
    # actual code is omitted for describing the template

def main():
  try:
    fh = open("service_Bench", 'rb')
    service = pickle.load(open("service_Bench"))
    service.run()
    pickle.dump(service,open("service_Bench",'wb'))
  except IOError:
    service = ServiceBench()
    service.run()
    pickle.dump(service,open("service_Bench",'wb'))
```

```
main()
```

5.3 Behaviour of Tuple Operations

We generate a seperate class for each intermediate tuple type to make the following operations easily implemented.

5.3.1 Comparation

When we compare two tuple variables, we omit the comparation of the variables' types. We compare each field of two variables to check whether the type and the value are the same. This makes the comparation of intermediate tuple variables compatible with normal tuple variables.

We implement two comparation operators (== and !=) by overloading the these operators in Python.

5.3.2 Assignment

When we make assignment of two tuple variables, we assign each field of two variables. This makes the assignment of intermediate tuple variables compatible with normal tuple variables.

We implement assignment by defining 'def assign(self, other)' method in each session class.

5.3.3 Keep/KeepMany

We didn't add new method for these two operators. We just use assignment method of the LHS tuple variable by sending all necessary values got from RHS.

5.3.4 Remove/RemoveMany

The same thing as the above

5.3.5 Join

We also implement this by using tuple assignment but we generate an intermediate tuple instance by sending all necessary values got from both LHS and RHS.

5.3.6 Initializaion

We implement this by creating an intermediate tuple instance by sending all necessary values got from the expression.

5.4 State Persistency

We use 'pickle' library of Python to implement both global and local state persistency. In this way, we don't need to deal with details of how to dump and load states. The 'pickle' technique (serializaion/unserialization) helps us to implement our class hierarchy design shown above in a nice and easy way.

When a generated CGI program runs, two types of files will be generated for state persistency: a file for the service and a file for each session.

5.5 Embedded Virtual Machine

Since Python doesn't support 'goto' and has strict requirement for code indentation, in order to emulate sequential threading, we designed an Embedded Virtual Machine mechanism described as follows.

The idea is we don't let the statement in generated script be executed directly, since otherwise we would have few control of where and when to stop and restart. So in each session, when we generate codes for each statement ('compountstm part in grammar'), we create a 'Instruction' object to store the statement string, its type and the offset (if needed), and insert it into a list called 'instructionStack'. When the generated CGI script starts running and enters a session, we read an instruction from the instructionStack and execute it as an atomic operation (using 'exec' method of Python) one by one. Also, according to the offset (positive means moving forward and negetive for moving back) of an instruction (only branching statement has non-zero offset) and the result of condition checking, the execution will jump back and forth. A program counter will keep the position of the next instruction for execution. When a show statement causes the program to stop, all local states including program counter will be store in a file; then when a form submission causes the program resume, all information are restored and the execution restart from last point. It is called 'Embedded' since it is not an independent process but embedded into each session (a Python class). Here is an example of the instructionStack-based execution for the following WIG program.

Embedded VM:

```
  def getNextInstruction(self):
    if self.pc < len(self.instructionStack):
      return self.instructionStack[self.pc]
    else:
      return None

  def run(self):
    while(True):
      nextInstruction = self.getNextInstruction()
      if nextInstruction == None:
        return
      if (nextInstruction.type == "if") or (nextInstruction.type == "ifelse") or (nextInstruction.type == "while"):
        exec str(nextInstruction.stm)
        if self._condition_flag == True:
          self.pc = self.pc + 1
        else:
          self.pc = self.pc + nextInstruction.offset
      if nextInstruction.type == "other":
        exec str(nextInstruction.stm)
        self.pc = self.pc + 1
      if nextInstruction.type == "return":
        self.pc = self.pc + 1
        return
      if nextInstruction.type == "goto":
        self.pc = self.pc + nextInstruction.offset
```

WIG program:

```
pc = 0;
while (pc < 2){
 pc = pc + 1;
 show Card_Info receive[s1=sender];
}
exit plug Card_Gen[greeting=s1];
```

$Sequence$	$Instruction$	$Type$	$Offset$
1	$"self.GetInfo_pc = 0"$	$other$	0
2	$"self._condition_flag = (self.GetInfo_pc < 2)"$	$while$	8
3	$"self.GetInfo_pc = self.GetInfo_pc + 1"$	$other$	0
4	$"self.service.makeFormHead(self.sessionid)"$	$other$	0
5	$"self.service.generateHtmlCard_Info()"$	$other$	0
6	$"self.service.makeFormTail()"$	$other$	0
7	$"return"$	$return$	0
8	$"self.getFields(["self.GetInfo_s1 = self.getvalue('sender')")"])"$	$other$	0
9	$"goto"$	$goto$	-7
10	$"print'Content - type : text/html'"$	$return$	0
11	$"self.service.generateHtmlCard_Gen(self.GetInfo_s1)"$	$return$	0
12	$"return"$	$return$	0

In the first run, the exectuion stopped at No.7 instruction; in the second run, it restarted from No.8 and jump back to No.2 when reached No.10; and repeat this process until '$self.GetInfo_pc < 2$' wasn't satisfied anymore and moved forward to No.11 and finished all execution No.12. Note, the 'goto' instruction is not an Python expression but our own tag. There are five types of such instruction described as follows:

$InstructionType$	$Workwith$	$OffsetType$
if	N/A	$positive$
$ifelse$	$goto$	$positive$
$while$	$goto$	$positive$
$goto$	$ifelse$	$positive$
$goto$	$while$	$negative$
$return$	N/A	0
$other$	N/A	0

5.6 Algorithm

The key point to make correct execution of Embedded VM is to generate the correct sequence of instructions and set the accurate offset for branching

We need to distinguish statement node and expression node when walking through AST in session scope. For session node we don't generate 'Instrucion' for it but just assemble the code string using children nodes; until we reach a statement node, we create a 'Instruction' node and insert it into the instrution list at the proper position. Since we are walking in a depthfirst way, children statements (they are visited first) will be added into list before parent statements (all compound statements). At this time, we need to insert the new instruction at some middle point in the list, e.g. 'if', 'ifelse' and 'while'. Now the key issue is to calculate the accute position of insertion and the offset set for the inserting instruction. we use a global counter to help us remember the sequence of current statement in a execution. More precisely, when we walk into a statement node, we make a footprint of the counter (i.e. store the counter number in a map using the node as the key, and we continue walking deeper; then when we walk back, we check our previous footprint (i.e. retrieve the counter form the map) and use it as the base with counter info from children statement to do all calculation; and last, update new counter info and keep it for parent nodes using in the future.

Algorithm example for 'While' statement:

```
inAWhileStm(node):
 // we will add two instructions later
 counter += 2
```

```
 // footprint
 map.put(node, counter)

outAWhileStm(node);
 // find previous footprint
 startPos = map.get(node)
 // get info of how many children instruction has been added
 endPos = map.get(node.childStm)
 // insert 'while' instruction in front of the list with an positive offset
 instructionList.insert(listlength-(endPos-startPos), 'while', (endPos - startPos + 2))
 // insert 'goto' instruction at the end of the list with an negetive offset
 instructionList.insert(listlength, 'goto', -(endPos-startPos+1))
 // reset footprint to reflect all changes under this node in AST
 map.put(node, counter)
```

5.7 Runtime System

Web Server with Python Scripts support.

5.8 Sample Code

Show the complete code generated for the service `template.wig`.

5.9 Testing

We've successfully tested nine of ten benchmark programs of this year and one benchmark of 1999 (tupletorture.wig to test tuple operations thoroughly). Only the benchmark of group2 (supercounter.wig) failed. We are still trying to solve it.

Another problem affected our progress is that the SOCS webserver is really unstable these days. It crashed from time to time (sometime it was down during a whole night) which made our debugging and testing very difficult complete in time.

However, we found a bug for the benchmark of group2 which has a const string escaping problem as follows:

Wrong syntax (the string of src attribute of img tag should be escaped by quotation):

```
string getNumber(int num, int type) {
  return "<img src=../images/counter/type"+ itoa(type) +"/"+ itoa(num) +".gif>";
}
```

Correct syntax:

```
string getNumber(int num, int type) {
  return "<img src=\"../images/counter/type"+ itoa(type) +"/"+ itoa(num) +".gif\">";
}
```

6 Availability and Group Dynamics

6.1 Manual

java -classpath wig03.jar Main yourprogram.wig

6.2 Demo Site

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

1. Group1
   - http://www.cs.mcgill.ca/ xsun16/cgi-bin/Hangman01.py?Play
2. Group2
   - http://www.cs.mcgill.ca/ xsun16/cgi-bin/Supercounter.py?ShowCounter
   - http://www.cs.mcgill.ca/ xsun16/cgi-bin/Supercounter.py?SetCounter
   - http://www.cs.mcgill.ca/ xsun16/cgi-bin/Supercounter.py?SetType
   - http://www.cs.mcgill.ca/ xsun16/cgi-bin/Supercounter.py?SetDigits
   - http://www.cs.mcgill.ca/ xsun16/cgi-bin/Supercounter.py?ResetCounter
3. Group3
   - http://www.cs.mcgill.ca/x̃sun16/cgi-bin/Postcard.py?GetInfo
4. Group4
   - http://www.cs.mcgill.ca/x̃sun16/cgi-bin/Test.py?calculate
5. Group5
   - http://www.cs.mcgill.ca/x̃sun16/cgi-bin/Wigverify.py?Authenticate
6. Group6
   - http://www.cs.mcgill.ca/x̃sun16/cgi-bin/Colors.py?ShowUsers
7. Group7
   - http://www.cs.mcgill.ca/x̃sun16/cgi-bin/Group07_wigservice.py?order
8. Group8
   - http://www.cs.mcgill.ca/x̃sun16/cgi-bin/Thr.py?calc
9. Group9
   - http://www.cs.mcgill.ca/x̃sun16/cgi-bin/Hangman09.py?Play
10. Group10
    - http://www.cs.mcgill.ca/x̃sun16/cgi-bin/Tupletorture.py?test1
    - http://www.cs.mcgill.ca/x̃sun16/cgi-bin/Tupletorture.py?test2

6.3 Division of Group Duties

All parts of the project was done concurrently by both of us together to avoid inconsistencies.