COMS W4115 Final Report
CSSLang

Wayne Yang (wy2149@columbia.edu)

August 13, 2010

Contents

1 Introduction

In the past few years, there has been a growing trend of building web sites that comply with web standards to ensure greater accessibility, usability, and interoperability of web pages. These web standards commonly refer to Hypertex Markup Language (HTML), Cascading Style Sheets (CSS), and ECMAScript (or Javascript). CSS is a style sheet language primarily used to describe the presentations of web pages [**?**]. It enables the separation of presentation from the semantic or structure of web pages and allows web designers to reuse style sheets across web pages [**?**].

Throughout this document, the term "CSS" refers to the the CSS Level 2 Revision 1 (CSS 2.1) specification by W3C. Although CSS can also be applied to XML documents, this project focuses on CSS in the HTML context.

This project is to design a language CSSLang that adds features typically seen in programming languages to CSS.

1.1 Motivations

Currently, CSS has no notion of variables which makes it diffcult for users to reuse style properties or create CSS rule sets that have parameterized property values. For example, to specify similar border and margin style for two class selectors *header* and *footer*, one has to duplicate style properties with slight variations in values:

```
#header {
   border-width: 1px;
   margin-width: 1px;
}
#footer {
   border-width: 5px;
   margin-width: 5px;
}
```

In addition, CSS lacks support for expressions in the property value such as the following:

```
#header {
   border-width: 5in + 3cm;
}
```

Inpsired by *LESS* and *Sass* which attempts to address these shortcomings of CSS, CSSLang implements some of the features and syntax of these languages.

2 Language Tutorial

At the top level of the source tree of **CSSLang**, type "make" on the command-line to compile all the source files. This should create an executable "csslang" which takes data from standard input.

Run "./testall.sh". The shell script runs all the test cases in the tests directory and outputs the following:

```
#######  Testing test-advance
PASS

#######  Testing test-arith
PASS

#######  Testing test-basic
PASS
```

2.1 Rule sets

Create a text file "foo" with the following content:

```
a { color: #16387c; }
a:visited { color: #5b73a3; }
a:visited.y-link-2 { color: #5b73a3; }

.clearfix:after {
content: ".";
display: block;
clear: both;
visibility: hidden;
line-height: 0;
height: 0;
}
```

Then run the command "./csslang ¡ foo" which will produce the following output:

```
a {
    color: #16387c;
}
a:visited {
    color: #5b73a3;
}
a:visited.y-link-2 {
```

```
    color: #5b73a3;
}
.clearfix :after {
    content: ".";
    display: block;
    clear: both;
    visibility: hidden;
    line-height: 0;
    height: 0;
}
```

csslang checks the syntax of the input data and formats the input into a canonical form. If the input is not a well-formed style sheet, then csslang return a parser error with the position of the offending token. For instance, the input below:

```
foobar {
    text-color: red
    background-color: blue;
}
```

will result in the error message "Parsing.Parse_error: line=3 cnum=20 token=[background-color]"

2.2 Variables

Varibles in **CSSLang** can be declared as follows:

```
$color = #FFFFFF;
$word = "hello!";
$greeting = 'Hello there';
$number = 123;
```

Variables can be assigned arithmetic expressions involving numbers that might have units:

```
/* foo is 17px */
$foo = 2px + 3px * 5px;
/* bar is 2cm */
$bar = 1cm + 10mm;
```

Variables can be used in a rule set as follows:

```
$border = 5px;

div#menu {
   border-width: $border;
}
```

which would be compiled into

```
$border = 5px;

div#menu {
   border-width: 5px;
}
```

2.3 Templates

Templates in **CSSLang** are essentially reusable rule sets with parameters. One can define a template as:

```
@set_space ($width) {
    margin-left: $width * 2px;
    margin-right: $width * 2px;
    padding-left: $width;
    padding-right: $width;
}
```

The set_space template can be included in a rule set:

```
div#menu {
    color: red;
    text-align: right;
    @set_space(2px);
}
```

which can be compiled into:

```
div#menu {
    color: red;
    text-align: right;
    margin-left: 4px;
    margin-right: 4px;
    padding-left: 2px;
    padding-right: 2px;
}
```

3 Language Reference Manual

3.1 Lexical Conventions

A CSSLang source file is a text file that contains zero or more of variables declarations, rule sets, and templates. The CSSLang compiler transforms a source file into a valid CSS file

by evaluating expressions and carrying out variable and template expansions. Any valid **CSSLang** source file can be compiled into a valid CSS 2.1 file, but not vice versa.

3.1.1 Tokens

There are four classes of tokens: identifiers, values, and operators.

3.1.2 White space

Characters "space" (U+0020), "tab" (U+0009), "line feed" (U+000A), "carriage return" (U+000D), and "form feed" (U+000C) are considered "white space" in **CSSLang**. White space and comments are ignored unless they separate tokens.

3.1.3 Comments

The characters /* introduce a comment and the characters */ terminate a comment. Comments do not nest.

3.1.4 Identifiers

CSSLang supports a subset of *identifiers* in CSS specification 2.1. In **CSSLang**, *identifiers* are sequences of characters [a-zA-Z], digits [0-9], hyphen (-), and the underscore (_). They can not start with a digit, or a hyphten followed by a digit. *identifiers* can appear as variable names, template names, property names, and selectors.

3.2 Selectors

In CSS, a selector is a pattern matching rule which determines which elements to apply the style rule to. The version of selector syntax implemented by **CSSLang** can be described in the following Yacc-like grammar:

```
selector_list:
    | simple_selector
    | selector_list COMMA simple_selector

simple_selector:
    element_name
    | element_name attrib_selector_list
    | attrib_selector_list

element_name:
    ID

attrib_selector_list:
```

```
    attrib_selector
    | attrib_selector_list attrib_selector

attrib_selector:
    HASHID
    | PERIOD ID
    | attrib
    | pseudo

pseudo:
    | COLON ID

attrib:
    LBRACKET ID attrib_match_op attrib_match_val RBRACKET
    | LBRACKET ID RBRACKET

attrib_match_op:
    ASSIGN
    | INCLUDES
    | DASHMATCH

attrib_match_val:
    ID
    | SSTRING
    | DSTRING
```

3.3 Rule sets

A rule set is a list of selectors followed by a declaration block. A declaration block begins with a left curly brace ({) and ends with a matching right curly brace (}). A declaration block can contain zero or more statements. An statement is either a property declaration or a template. A property declaration is a pair of property name and an expression which will be evaluated to a literal. The property name and value are separated by a colon (:) and terminated by a semicolon (;). A property name is an identifier. Literals and expressions will be described in details in later sections.

The following is the format of a rule set:

```
ruleset_decl:
    selector_list LBRACE vdecl_list stmt_list RBRACE

prop_decl:
    | prop COLON value
```

```
prop:
    | ID

vdecl:
    | DOLLAR ID ASSIGN expr SEMI

vdecl_list:
    /* nothing */
  | vdecl_list vdecl


stmt_list:
  /* empty */
  | stmt_list stmt SEMI

stmt:
    prop_decl
    | ATRULE ID LPAREN actuals_opt RPAREN

actuals_opt:
    /* empty */
    | actuals_list

actuals_list:
    expr
  | actuals_list COMMA expr
```

The following is an example of a rule set:

```
div.header {
    float: left;
    color: #FFFFFF;
}
```

3.4 Literals

CSSLang supports four main types of literals: numbers, single-quoted strings, double-quoted strings, and unquoted strings. Literals can appear on the right-hand side of variable assignment.

3.4.1 Numbers

Numbers can have integer values or floating values. An integer constant consists of a sequence of digits and is taken to be decimal. A floating constant consists of an integer part, a decimal point, and a fraction part. Either the integer part or the fraction part (not both) may be missing. Numbers may be preceded by a "-" or "+" to indicate the sign. -0 is equivalent to 0. Numbers may be immediately followed by a unit. There are two types of units: relative and absolute. Relative units are **em**, **ex**, and **px**. Absolute units are **in**, **cm**, **mm**, **pt**, and **pc**. For details on these length units, please refer to the CSS 2.1 specification.

Only numbers of the same relative unit can appear as operands of an arithmetic operator. Conversion happens when numbers in different absolute units appear in an arithmetic expression. The unit of the left operand will be used as the base unit for conversion. The table below is used by CSSLang to perform conversions between numbers in different absolute units:

from	to	to/from
in	cm	2.54
in	mm	25.4
in	pt	72.0
in	pc	6.0
cm	in	0.3937
cm	mm	10.0
cm	pt	28.3465
cm	pc	2.3622
mm	cm	0.1
mm	in	0.03937
mm	pt	2.8347
mm	pc	0.23622
pt	in	0.01388
pt	cm	0.03527
pt	mm	0.35277
pt	pc	0.08333
pc	in	0.16667
pc	cm	0.42333
pc	mm	4.23333
pc	pt	12.0

3.4.2 Strings

String literals can be specified with double quotes ("), single quotes ('), or without quotes. If string literals are quoted, they must have matching quotes.

3.5 Operators

3.5.1 Parentheses ()

Parentheses are used to group expressions. A parenthesized expression is a primary expression whose type and value are identical to those of the original expression.

3.5.2 Precedence & Associativity

The table below summarizes the rules for precedence and associativity of all operators. Operators on the same line have the same precedence; rows are in decreasing precedence:

OPERATORS	ASSOCIATIVITY
()	left to right
!	right to left
* /	left to right
+ −	left to right

3.6 Expressions

Expressions are variables, strings, numbers or expressions in parentheses. Expressions are evaluated according to the associativity and precedence of operators involved. Parentheses can be used to change the order of evaluations in expressions.

Primary expressions are identifiers, values, or expressions in parentheses.

3.7 Variables

Variables are identifiers preceded with a dollar sign ($). Variables can be declared inside or outside of a rule set. When a variable is declared, a value must be given at the same time.

```
$variable-name = expressions semicolon
```

3.7.1 Variable scope

Varaibles in **CSSLang** are statically scoped. Variables declared outside of rule sets are global. Variables declared inside a rule set are only visible within the rule set. The values of variables persist after the variable declarations until the end of a file or the end of a rule set. Variables can be used as a property value in rule sets and templates, and on the right-hand side of assignments are substituted with their values. The following is an source file that illustrates the scoping rules in **CSSLang**:

```
$foo = beagle
$bar = bulldog
```

```
A {
   $bar = labrador
   dog1: $foo
   dog2: $bar
}
```

This source file is compiled into the following:

```
A {
   $bar = labrador
   dog1: beagle
   dog2: labrador
}
```

4 Templates

CSSLang adds support for directives that allow rule templates.

The templates can take parameters which can be expressions. A template is declared outside of rule sets with an at sign ('@') followed by an identifier follwed by parentheses and a declaration block. The parentheses can contain a comma-separated list of parameters in variable notation:

```
@template_name($param1, $param2, ...) {
   properties
}
```

A template can be used inside a declaration block of rule sets or templates by using the at sign ('@') followed by the template name and its parameters in parentheses.

```
selector {
   @template_name($param1, $param2);
}
```

A template inclusion inside a rule set is considered a statement and it should be terminated by a semicolon.

5 Project Plan

For CVN students, this project represents an opportunity for self-directed learning and managing a project from start to end. This also means that there is no communicaton overhead among team members. But at the same time, one does not have the luxury of sharing workload or working collaboratively with others. So I've chosen to work on a language relating to the area I am most familiar with: web standards and technologies.

5.1 CSS 2.1 Specification

Once the features of the language have been identified, the next step is to review the CSS Level 2 Revision 1 Specification in great details. Understanding the CSS specification is crucial as the project goals are to build a CSS 2.1 parser and to extend CSS with features borrowed from scripting languages. During the study of the CSS specification, I've decided to keep a list of CSS syntax to exclude from **CSSLang** in order to keep the project scope manageable.

5.2 Ocamllex, Ocamlyacc, and microc

In order to start the actual design and implementation of **CSSLang**, I've devoted a considerable time in studying the implementation and source code structure of the *microc* language. Besides the coverage of microc in lectures, I've also found the Ocamllex and Ocamlyacc Tutorial by SooHyoung Oh to be very valuable. After studying Ocamllex, Ocamlyacc, and the microc language, I was able to start the language implementation in the order of scanner, parser, data structures, and the interpreter.

5.3 Source control software

In order to keep a history of different revisions of various project files, the first thing before writing any lines of code is to set up a Subversion source repository. Subversion was chosen as the source version control repository because of its widespread usage and intuitive user interface. Although there are several web sites that offer free Subversion access, it is easier and more efficient to set up my own Subversion server. The guideline I use for development is to commit incremental and meaningful changes as soon as they are tested. This allows me to easily identify new changes that break existing features and to roll back the source files to a previous known state if necessary.

5.4 Software development environment

- Ubuntu 9.10 with Linux kernel 2.6.31-22
- Objective Caml version 3.11.2
- Subversion 1.6.5
- GNU Make 3.81
- VIM - Vi IMproved 7.2
- pdfTeXk, Version 3.141592-1.40.3 - An extended version of TeX

5.5 Project log

6/2/2010-6/4/2010	Research and brainstorm
6/6/2010	Project proposal
6/23/2010	First draft of language reference manual
6/25/2010	Language reference manual
7/5/2010	Language reference manual revision 1
7/14/2010	Set up subversion source repository
7/26/2010	Create Makefile and stub of scanner and parser files
8/5/2010	First version of scanner
8/8/2010	First version of parser
8/10/2010	First version of ast , interpreter, and main program
8/11/2010	Test scripts
8/12/2010	Final report

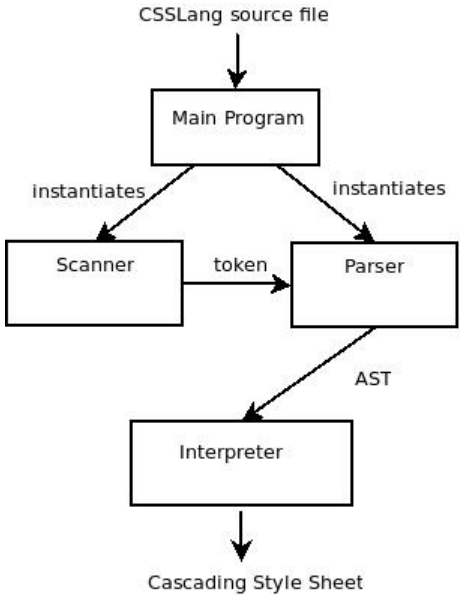
5.6 Features not implemented

The following is a list of features planned originally but are not implemented:

- Boolean literals
- Relational, equality, and inequality operators
- Negation operator
- Control structures such as If/else

6 Architectural Design

The architecture of CSSLang follows closely that of microc. The diagram below shows the major components of this project:



6.1 Main program

The main program, csslang.ml, is responsible for combining the lexer (scanner), parser, and the interpreter. It creates a lex buffer from the standard input and does the top-level error handling of exceptions.

6.2 Scanner

Scanner is the lexical analyzer of the system. It is created by running Ocamllex on a set of regular expressions and semantic actions defined in scanner.mll. The scanner has an entry function *Scanner.csslang_token* that returns a token defined in the parser module.

6.3 Parser

The parser component is generated by running Ocamlyacc on a set of grammars and its semantic actions. The entry point function of parser is the name of the start symbol, *Parser.program*. This function returns the abstract syntax tree (AST) that represents a **CSSLang** program.

6.4 Interpreter

The interpreter takes the AST returned from the parser as input. The AST consists of a tuple of a list of variable declarations, a list of rule sets, and a list of templates. The interpreter walks the these data structures and creates the symbol tables for global and local variables as well as for the templates. It then prints all the rule sets by performing the variables and templates substitutions and evaluates any expressions. The interpreter

checks for errors such as undefined variables or templates and makes sure templates are called with the correct actual parameters.

7 Test Plan

The scanner and parser components were mostly developed independently from the rest of the programs. They were written initially to write the string representation to the standard output for testing purposes. The testings for these two components were done in an ad-hoc manner since they would also be tested in the end-to-end test cases. This also gives the benefit of speeding up the initial development cycle.

7.1 Source programs and generated programs

The following is an example of CSSLang source code:

```
$default_margin = 2px;
$default_padding = 5px;

/*
Set the space around a html element
*/
@set_space ($margin_size, $padding_size) {
    margin-left: $margin_size;
    margin-right: $margin_size;
    padding-left: $padding_size;
    padding-right: $padding_size;
}

/*
Set the foreground and background color
*/
@set_color ($text_color, $background_color) {
    color: $text_color;
    background-color: $background_color;
}

div#login {
    @set_space($default_margin, $default_padding);
    @set_color(white, black);

}
```

```
button.submit {
    $margin = $default_margin * 2px;
    $padding = $default_padding * 5px ;
    @set_space($margin, $padding);
}

h1.warning {
    $margin = 5in;
    @set_space($margin, $default_padding);
}
```

The generated output is as follows:

```
div#login {
    margin-left: 2px;
    margin-right: 2px;
    padding-left: 5px;
    padding-right: 5px;
    color: white;
    background-color: black;
}
button.submit {
    margin-left: 4px;
    margin-right: 4px;
    padding-left: 25px;
    padding-right: 25px;
}
h1.warning {
    margin-left: 5in;
    margin-right: 5in;
    padding-left: 5px;
    padding-right: 5px;
}
```

7.2 Test cases

7.2.1 Test on various selector syntax

```
/*
Different syntax of selector:
    - element name
    - class selector (e.g. .abc)
    - ID selector (e.g. #input)
    - element name followed by class|ID selector
```

```
    - pseudo-class selector (e.g. :link)
    - A combination of the above selector
*/

A {
}

input123 {
    display: inline;
}

.abc {
}

#input {

}

:link {
    color: red;
}

A, button {
}

A:visited, button.submit {
    text-align: right;
}

body, #foo {
    top: 12px;
}

foo_bar, :visited {
    border-color: red;
}
```

7.2.2 Test on various arithmetic expressions

```
/*
Test arithmetic expressions
*/
$test1 = 5pc + (1in + 4mm) + 4pt;
```

```
$test2 = 1 + 2 * 3 + 4;
$test3 = 10px - (5px - 6px);
$test4 = 1/0;
$test5 = 100mm / 1cm * 30mm;
$test6 = 5in - 4cm * 3pt / 2pc;

arithmetics {
    test1: $test1;
    test2: $test2;
    test3: $test3;
    test4: $test4;
    test5: $test5;
    test6: $test6;
}
```

7.2.3 Test on templates

```
/**
 * - Test template definition and inclusion
 * - Test use of expressions in variable assignment
 * - Test implicit unit conversion in expression involving absolute length unit
 */
$abc = 2cm + 12mm;
$font_size = 2em * 10em;
$align = center;
$white_color = #FFFFFF;

.cta-btn-ext:visited {
    $space = 5pc;

    color: $white_color;
    font-weight: bold;
    font-size: $font_size;
    text-decoration: none;
    text-align: $align;
    border: $abc;
    top: $space;
    left: $space;
}

/**
    A template that sets area around a HTML element container
*/
```

```
@set_space ($bar) {
    margin-left: 1px;
    margin-right: 1px;
    padding-left: $bar;
    padding-right: $bar;
}


submit {
    $color = green;
    $list-style = none;
    text-align: $align;
    @set_space(10px + 20px);
    text-color: $color;
}
```

7.3 Test program

The test automation is implemented as a simple a Bash shell script **testall.sh** which looks at all the test files under the *tests* directory. All the test files have prefix *test*. The suffix '.in' is used for test input files. The suffix '.out' is used for the expected output files. When the **testall.sh** runs, it iterates through all the test input files and generates intermediate files with suffix '.generated' that contains the output of running the **csslang** executable on the test input files. It then uses *diff* to examine if there's any difference between the *.generated files and *.out files. Attached below is the test automation program:

```
#!/bin/bash
# Test program output is stored in {testCase}.generated
# The diff between program output and expected output is stored
# in {testCase}.diff

CSSLANG="./csslang"
TESTDIR="tests"
files=‘ls $TESTDIR/test-*.in‘

for file in $files
do
    basename=‘basename $file‘;
    basename=‘echo $basename | sed ’s/.in$//’‘
    echo "#######  Testing $basename";
    $CSSLANG < $file > $TESTDIR/${basename}.generated;
    if diff -b $TESTDIR/${basename}.out $TESTDIR/${basename}.generated
        > $TESTDIR/${basename}.diff; then
```

```
        rm $TESTDIR/${basename}.diff $TESTDIR/${basename}.generated;
        echo "PASS";
    else
        echo "FAIL";
    fi
    echo;

done
```

8 Lessons Learned

The compiler project has been a tremendous learning experience for me. I was able to learn the basic building blocks of language design and how OCaml's type system allows the implementation to be more compact. One of the challenges is the time management of this project because of having to work on all aspects of the project.

Another lesson learned is the use of test files. Without enough test files, it was hard to make changes in the interpreter component without affecting other features. Things would have gone a lot smoother if I have created test automation and the skeletons of all the components in the system first, then gradually add features and corresponding test cases.

9 Appendix

9.1 CSS 2.1 syntax not implemented in CSSLang

CSSLang does not implement some of the syntax of CSS 2.1. The listing of syntax is as follows:

- does not support SGML comment delimitors
- not support !important syntax
- no space in the url expression and must be a quoted string inside
- no function as expression
- does not support "shorthanded" notion of property value that is delimited by space
- does not support unicode characters
- smaller set of characters allowed for identifiers
- does not support various at rules such as @import or @charset

- does not support RGB color value such as rgb(100
- does not support descendant selectors, child selectors, adjacent selectors or attribute selectors

9.2 Source files

9.2.1 Scanner - scanner.mll

```
(*
scanner.mll
-----------
This file specifies the regular expressions and the semantic actions for the lexical
*)

(* header *)
{
    open Parser
    open Printf
    open Char
}

(* definitions section *)
let alphabet = ['a'-'z''A'-'Z']
let digit = ['0'-'9']
let start_namechar = alphabet | '_' | '-'
let name_char = alphabet | '-' | '_' | digit
let name = name_char+
let ident = start_namechar name_char*
let decimal_point = '.'
let num = digit+ | digit* decimal_point digit+
let at = '@'
let nl = '\n' | "\r\n" | '\r'
let squote = "'"
let dquote = '"'
let single_quote_str = squote ([^'\n' '\r' '\''] | '\\' nl )* squote
let double_quote_str = dquote ([^'\n' '\r' '"'] | '\\' nl )* dquote
let quoted_str = single_quote_str | double_quote_str
let noquote_str = [^'\n' '\r' '\'' '"']+
let space = [' ' '\t' '\r']
let url = "url(" (quoted_str | noquote_str) ")"
let hex = ['0'-'9''a'-'f''A'-'F']
let hex_color = "#" (hex hex hex | hex hex hex hex hex hex)

(* rules section *)
rule csslang_token = parse
| space { csslang_token lexbuf } (* Whitespace *)
| nl { Lexing.new_line lexbuf; csslang_token lexbuf } (* update position *)
```

```
| url as s { URI(s) }
| "em" as s { EM(s) }
| "ex" as s { EX(s) }
| "px" as s { PX(s) }
| "cm" as s { CM(s) }
| "mm" as s { MM(s) }
| "in" as s { IN(s) }
| "pt" as s { PT(s) }
| "pc" as s { PC(s) }
| hex_color as s { HEXCOLOR(s) }
| "$"      { DOLLAR }
| "/*"          { comment lexbuf } (* Comments *)
| "(" as s { LPAREN(Char.escaped s) }
| ")" as s { RPAREN(Char.escaped s) }
| "{"       { LBRACE }
| "}"       { RBRACE }
| "[" as s { LBRACKET(Char.escaped s) }
| "]" as s { RBRACKET(Char.escaped s) }
| "~=" as s { INCLUDES(s) }
| "|=" as s { DASHMATCH(s) }
| ";"       { SEMI }
| "+" as s { PLUS(Char.escaped s) }
| "-" as s { MINUS(Char.escaped s) }
| "*" as s { TIMES(Char.escaped s) }
| "/" as s { DIVIDE(Char.escaped s) }
| "." as s { PERIOD(Char.escaped s); }
| ":" as s { COLON(Char.escaped s) }
| '=' as s { ASSIGN(Char.escaped s) }
| ','      { COMMA }
| "=="     { EQ }
| "!="     { NEQ }
| '<'      { LT }
| "<="     { LEQ }
| ">"      { GT }
| ">="     { GEQ }
| ident as s { ID(s) }
| '@' { ATRULE }
| "#" name as s      { HASHID(s) }
| single_quote_str as s { SSTRING(s) }
| double_quote_str as s { DSTRING(s) }
| num as s  { NUMBER(s) }
| '%' as s { PERCENT(Char.escaped s) }
```

```
| _ as char { raise (Failure("illegal character " ^ Char.escaped char)) }
| eof { EOF }
and comment = parse
  "*/" { csslang_token lexbuf }
| _    { comment lexbuf }
```

9.2.2 Parser - parser.mly

```
%{
    (* This is the file that specifies the gramma and the
       semantic actions to create the AST
    *)
    open Ast
    open Printf
    open List
    let list2str s e = s ^ e;;
    let list2str2 s e =
        match s with
            ""  -> e
            | _   -> s ^ " " ^ e
    let list2str3 s e =
        match s with
            "" -> e
            | _ -> s ^ ", " ^ e
%}

%token SEMI LBRACE RBRACE COMMA SPACE DOLLAR ATRULE
%token <string> PLUS MINUS TIMES DIVIDE
%token EQ NEQ LT LEQ GT GEQ
%token <string> NUMBER
%token <string> ID HASHID SSTRING DSTRING URI PERIOD
%token <string> COLON LBRACKET RBRACKET ASSIGN INCLUDES DASHMATCH
%token <string> LPAREN RPAREN PERCENT
%token <string> EM EX PX CM MM IN PT PC HEXCOLOR
%token EOF

%nonassoc NOELSE
%nonassoc ELSE

%left ASSIGN
%left EQ NEQ
%left LT GT LEQ GEQ
%left PLUS MINUS
```

```
%left TIMES DIVIDE

%start program
%type <Ast.program> program

%%

program:
    /* empty */ { [], [], [] }
    | program vdecl {
        let (v, r, d) = $1 in
        ($2 :: v, r, d) }
    | program ruleset_decl {
        let (v, r, d) = $1 in
        (v, $2 :: r, d) }
    | program defdecl {
        let (v, r, d) = $1 in
        (v, r, $2 :: d) }

defdecl:
    ATRULE ID LPAREN formals_opt RPAREN LBRACE vdecl_list stmt_list RBRACE {{
        fname = $2;
        formals = $4;
        dlocals = $7;
        body = List.rev $8; }}

vdecl:
    | DOLLAR ID ASSIGN expr SEMI
        {  { vdecl_key = $2; vdecl_val = $4; }}

vdecl_list:
    /* nothing */    { [] }
  | vdecl_list vdecl { $2 :: $1 }


stmt_list:
  /* empty */  { [] }
  | stmt_list stmt SEMI { $2 :: $1 }

stmt:
    prop_decl { let (s, expr) = $1 in PropDecl(s, expr) }
    | ATRULE ID LPAREN actuals_opt RPAREN { IncludeDef($2, $4) }
```

```
actuals_opt:
    /* empty */ { [] }
    | actuals_list  { List.rev $1 }

actuals_list:
    expr                    { [$1] }
  | actuals_list COMMA expr { $3 :: $1 }

formals_opt:
  /* empty */ { [] }
  | formal_list   { List.rev $1 }

formal_list:
  DOLLAR ID                   { [ $2 ] }
  | formal_list COMMA DOLLAR ID { $4 :: $1 }


expr:
    literal { Literal($1) }
    | DOLLAR ID { Id($2) }
    | expr PLUS   expr { Binop($1, Add,   $3) }
    | expr MINUS  expr { Binop($1, Sub,   $3) }
    | expr TIMES  expr { Binop($1, Mult,  $3) }
    | expr DIVIDE expr { Binop($1, Div,   $3) }
    | LPAREN expr RPAREN { $2 }

literal:
    | ID {{
        bv_fval = 0.0;
        bv_sval = $1;
        bv_type = TYPE_NOQUOTE_STR;
        bv_unit = Nounit; }}

    | NUMBER type_unit {{
        bv_fval = (float_of_string $1);
        bv_sval = "";
        bv_type = TYPE_NUM;
        bv_unit = $2; }}
    | NUMBER {{
        bv_fval = (float_of_string $1);
        bv_sval = "";
        bv_type = TYPE_NUM;
        bv_unit = Nounit; }}
```

```
    | SSTRING {{
        bv_sval = $1;
        bv_fval = 0.0;
        bv_type = TYPE_SQUOTE_STR;
        bv_unit = Nounit; }}
    | DSTRING {{
        bv_sval = $1;
        bv_fval = 0.0;
        bv_type = TYPE_DQUOTE_STR;
        bv_unit = Nounit; }}
    | HEXCOLOR {{
        bv_sval = $1;
        bv_fval = 0.0;
        bv_type = TYPE_NOQUOTE_STR;
        bv_unit = Nounit; }}

type_unit:
    | PERCENT { Rellen($1) }
    | EM { Rellen($1) }
    | EX { Rellen($1) }
    | PX { Rellen($1) }
    | CM { Abslen($1) }
    | MM { Abslen($1) }
    | IN { Abslen($1) }
    | PT { Abslen($1) }
    | PC { Abslen($1) }


ruleset_decl:
    selector_list LBRACE vdecl_list stmt_list RBRACE {{
        selector = (List.fold_left list2str3 "" (List.rev $1));
        declaration = (List.rev $4);
        rlocals = $3; }}

prop_decl:
    | prop COLON expr { ($1, $3) }

prop:
    | ID { $1 }

value:
    | DOLLAR ID {{
        bv_sval = $2;
```

```
        bv_fval = 0.0;
        bv_type = TYPE_VAR;
        bv_unit = Nounit; }}
    | URI {{
        bv_sval = $1;
        bv_fval = 0.0;
        bv_type = TYPE_NOQUOTE_STR;
        bv_unit = Nounit; }}
    | NUMBER type_unit {{
        bv_sval = "";
        bv_fval = (float_of_string $1);
        bv_type = TYPE_NUM;
        bv_unit = $2; }}
    | NUMBER {{
        bv_sval = "";
        bv_fval = (float_of_string $1);
        bv_type = TYPE_NUM;
        bv_unit = Nounit; }}
    | SSTRING {{
        bv_sval = $1;
        bv_fval = 0.0;
        bv_type = TYPE_SQUOTE_STR;
        bv_unit = Nounit; }}
    | DSTRING {{
        bv_sval = $1;
        bv_fval = 0.0;
        bv_type = TYPE_DQUOTE_STR;
        bv_unit = Nounit; }}
    | ID {{
        bv_sval = $1;
        bv_fval = 0.0;
        bv_type = TYPE_NOQUOTE_STR;
        bv_unit = Nounit; }}
    | HEXCOLOR {{
        bv_sval = $1;
        bv_fval = 0.0;
        bv_type = TYPE_NOQUOTE_STR;
        bv_unit = Nounit; }}

selector_list:
    | simple_selector { [ $1 ] }
    | selector_list COMMA simple_selector { $3 :: $1 }
```

```
simple_selector:
    element_name                        { $1 }
    | element_name attrib_selector_list
        { $1 ^ (List.fold_left list2str "" (List.rev $2)) }
    | attrib_selector_list
        { List.fold_left list2str2 "" (List.rev $1) }

element_name:
    ID { $1 }

attrib_selector_list:
    attrib_selector { [ $1 ] }
    | attrib_selector_list attrib_selector { $2 :: $1 }

attrib_selector:
    HASHID { $1 }
    | PERIOD ID { $1 ^ $2 }
    | attrib { $1 }
    | pseudo { $1 }

pseudo:
    | COLON ID { $1 ^ $2 }

attrib:
    LBRACKET ID attrib_match_op attrib_match_val RBRACKET
        { $1 ^ $2 ^ $3 ^ $4 ^ $5 }
    | LBRACKET ID RBRACKET
        { $1 ^ $2 ^ $3 }

attrib_match_op:
    ASSIGN    { $1 }
    | INCLUDES  { $1 }
    | DASHMATCH { $1 }

attrib_match_val:
    ID          { $1 }
    | SSTRING    { $1 }
    | DSTRING    { $1 }
```

9.2.3 Interpreter - interpret.mly

```
(*
interpret.ml
```

```
-----------
This file implements the interpreter that walks the ast and evaluates
statements and expressions.
*)


open Ast
open Printf

module NameMap = Map.Make(struct
  type t = string
  let compare x y = Pervasives.compare x y
end)

let match_type a b =
    if (a.bv_type != b.bv_type) then
        raise (Failure ("Operand type mismatch"));;

let check_num a b =
    if (a.bv_type != TYPE_NUM || b.bv_type != TYPE_NUM) then
        raise (Failure ("Operand is not a number"));;

let match_num a b =
    (match_type a b);
    (check_num a b)

let convert_unit ufrom uto =
    match ufrom,uto with
        "in", "cm" -> 2.54
        | "in", "mm" -> 25.4
        | "in", "pt" -> 72.0
        | "in", "pc" -> 6.0
        | "cm", "in" -> 0.3937
        | "cm", "mm" -> 10.0
        | "cm", "pt" -> 28.3465
        | "cm", "pc" -> 2.3622
        | "mm", "cm" -> 0.1
        | "mm", "in" -> 0.03937
        | "mm", "pt" -> 2.8347
        | "mm", "pc" -> 0.23622
        | "pt", "in" -> 0.01388
        | "pt", "cm" -> 0.03527
        | "pt", "mm" -> 0.35277
```

```
        | "pt", "pc" -> 0.08333
        | "pc", "in" -> 0.16667
        | "pc", "cm" -> 0.42333
        | "pc", "mm" -> 4.23333
        | "pc", "pt" -> 12.0
        | "in", "in" -> 1.0
        | "cm", "cm" -> 1.0
        | "mm", "mm" -> 1.0
        | "pt", "pt" -> 1.0
        | "pc", "pc" -> 1.0
        | _, _ -> raise (Failure ("Can not convert units"))


let normalize_unit a b =
    (* check unit type *)
    match a.bv_unit, b.bv_unit with
        (* convert b to a's unit type *)
        Abslen(i), Abslen(j) ->
            let new_b = { b with bv_fval = (convert_unit j i) *. b.bv_fval} in
            a,new_b
        | Rellen(i), Rellen(j) ->
            if i = j then a, b
            else raise
                (Failure ("Invalid operation on different relative unit types"))
        | Nounit, Nounit -> a,b
        | _, _ -> raise (Failure ("Incompatible operand types"))


let string_of_unit u =
    match u with
        Abslen(i) -> i
        | Rellen(i) -> i
        | Nounit -> "";;

let trim c str =
    let n = String.length str in
    let str =
        if str.[n-1] = c then String.sub str 0 (n-1)
        else str in
    str

(* Evaluate an expression and return (value, updated environment) *)
let rec eval env = function
```

```
Literal(i) -> i, env
| Id(var) ->
    let locals, globals = env in
    if NameMap.mem var locals then
        (NameMap.find var locals), env
    else if NameMap.mem var globals then
        (NameMap.find var globals), env
    else raise (Failure ("undeclared identifier " ^ var))
| Binop(e1, op, e2) ->
    let v1, env = eval env e1 in
    let v2, env = eval env e2 in
    (* let boolean i = if i then 1 else 0 in *)
    (match op with
        Add ->
            let (v1, v2) = (normalize_unit v1 v2) in
            {
                bv_fval = v1.bv_fval +. v2.bv_fval;
                bv_type = TYPE_NUM;
                bv_unit = v1.bv_unit;
                bv_sval = "";
            }
        | Sub ->
            let (v1, v2) = (normalize_unit v1 v2) in
            {
                bv_fval = v1.bv_fval -. v2.bv_fval;
                bv_type = TYPE_NUM;
                bv_unit = v1.bv_unit;
                bv_sval = "";
            }
        | Mult ->
            let (v1, v2) = (normalize_unit v1 v2) in
            {
                bv_fval = v1.bv_fval *. v2.bv_fval;
                bv_type = TYPE_NUM;
                bv_unit = v1.bv_unit;
                bv_sval = "";
            }
        | Div ->
            let (v1, v2) = (normalize_unit v1 v2) in
            {
                bv_fval = v1.bv_fval /. v2.bv_fval;
                bv_type = TYPE_NUM;
                bv_unit = v1.bv_unit;
```

```
                    bv_sval = "";
                }), env

            (*
            | Equal -> boolean (v1 = v2)
            | Neq -> boolean (v1 != v2)
            | Less -> boolean (v1 < v2)
            | Leq -> boolean (v1 <= v2)
            | Greater -> boolean (v1 > v2)
            | Geq -> boolean (v1 >= v2)), env
            | Assign(var, e) ->
                let v, (locals, globals) = eval env e in
                    if NameMap.mem var locals then
                    v, (NameMap.add var v locals, globals)
                    else if NameMap.mem var globals then
                    v, (locals, NameMap.add var v globals)
                    else raise (Failure ("undeclared identifier " ^ var))
            *)
    | Id(id) ->
        {
            bv_fval = 0.0;
            bv_type = TYPE_NUM;
            bv_unit = Nounit;
            bv_sval = "";
        }, env
    | Assign(s, e) ->
        {
            bv_fval = 0.0;
            bv_type = TYPE_NUM;
            bv_unit = Nounit;
            bv_sval = "";
        }, env


let rec string_of_bval a env =
    match a.bv_type with
        TYPE_NUM ->
           (trim '.' (string_of_float a.bv_fval)) ^ (string_of_unit a.bv_unit), env
        | TYPE_SQUOTE_STR -> a.bv_sval, env
        | TYPE_DQUOTE_STR -> a.bv_sval, env
        | TYPE_NOQUOTE_STR -> a.bv_sval, env
        | TYPE_VAR ->
            let x = Id(a.bv_sval) in
```

```
            let (bval, env) = eval env x in
            string_of_bval bval env


let run (var_decls, ruleset_decls, def_decls) =

    (* helper function to populate a map with a var_decl list *)
    let load_vars_to_map env var_decls =
        let (locals, globals) = env in
        let get_eval env bvalue =
            let (v, _) = eval env bvalue in v
        in
        let varmap =
        List.fold_left
           (fun varmap decl -> NameMap.add decl.vdecl_key
               (get_eval (locals,globals) decl.vdecl_val) varmap)
           NameMap.empty var_decls
        in varmap
    in

    (* Populate global symbol table *)
    let globals = load_vars_to_map (NameMap.empty, NameMap.empty) var_decls in

    let includes = List.fold_left
        (fun def_decls def -> NameMap.add def.fname def def_decls)
        NameMap.empty def_decls
    in

    (*
    @return string, (NameMap, NameMap)
    *)
    let rec string_of_stmt stmt env =

        match (stmt) with
        PropDecl(vname, expr) ->
            let (bval, env) = (eval env expr) in
            let (str, env) = (string_of_bval bval env) in
            "\t" ^ vname ^ ": " ^ str ^ ";\n", env
        | IncludeDef(dname, actuals) ->
            let inc =
                try NameMap.find dname includes
                with Not_found -> raise (Failure ("undefined template " ^ dname))
            in
```

```
        (* evaluate the parameters passed in *)
        let actuals, env = List.fold_left
            (fun (actuals, env) actual ->
            let v, env = eval env actual in v :: actuals, env)
            ([], env) actuals
        in
        let actuals = List.rev actuals in

        (* bind actual values to formal arguments *)
        let tpl_locals =
            try List.fold_left2
                (fun locals formal actual -> NameMap.add formal actual locals)
                NameMap.empty inc.formals actuals
            with Invalid_argument(_) ->
            raise (Failure ("wrong number of arguments passed to " ^ dname))
        in

        (* evaluate statements in  template definition *)
        let tpl_env = (tpl_locals, snd env) in
        let str_list, _ = List.fold_left
            (fun (str_list, env) stmt ->
                let s, env = string_of_stmt stmt env in s :: str_list, env)
            ([], tpl_env) inc.body
        in
        let str = String.concat "" (List.rev str_list) in
        str, env

    | VarDecl(vdecl) ->
        let varname = vdecl.vdecl_key in
        let varval = vdecl.vdecl_val in
        let (bval , env) = eval env varval in

        (* add the var decl to the locals *)
        let (locals, globals) = env in

        let locals = NameMap.add varname bval locals in
        "", (locals, globals)
in

(* @param decl: stmt list *)
let string_of_declaration stmt_list env =
    let str_list, env = List.fold_left
        (fun (str_list, env) stmt ->
```

```
            let s, env = string_of_stmt stmt env in s :: str_list, env)
        ([], env) stmt_list
    in
    String.concat "" (List.rev str_list)
    (*
    List.fold_left (fun s stmt -> s ^ (string_of_stmt stmt env))
    *)
in

let string_of_ruleset ruleset =
    let locals = load_vars_to_map (NameMap.empty, globals) ruleset.rlocals in
    let env = (locals, globals) in
    (* debug
    printf "selector=%s\n" ruleset.selector;
    *)
    ruleset.selector ^ " {\n" ^
    (string_of_declaration ruleset.declaration env) ^ "}\n"
in

let string_of_program globals rulesets =
    String.concat "" (List.map string_of_ruleset (List.rev rulesets))
in

printf "%s" (string_of_program globals ruleset_decls)
```

9.2.4 Main program - csslang

```
(*
csslang.ml
----------
This is the main program of csslang.
It reads input from standard input and combines scanner, parser, and interpreter.
*)

open Ast
open Printf
open Printexc

let _ =
    let lexbuf = Lexing.from_channel stdin in
    try
        let program = Parser.program Scanner.csslang_token lexbuf in
            Interpret.run program;
```

```
            ();
            (*
            List.map (fun a -> printf "key=%s\n" a.vdecl_key) vars;
            let listing = string_of_program program in
            (printf "%s" listing);
            *)
    with
        | exn
            ->
               let curr = lexbuf.Lexing.lex_curr_p in
               let cnum = curr.Lexing.pos_cnum - curr.Lexing.pos_bol in
               let tok = Lexing.lexeme lexbuf in
               printf "%s: " (Printexc.to_string exn);
               printf "line=%d cnum=%d " curr.Lexing.pos_lnum cnum;
               printf "token=[%s]\n" tok;
               ();
```

9.2.5 Makefile

```
OBJS = parser.cmo scanner.cmo interpret.cmo csslang.cmo

csslang: $(OBJS)
ocamlc -o csslang $(OBJS)

scanner.ml : scanner.mll
ocamllex scanner.mll

parser.ml parser.mli : parser.mly
ocamlyacc -v parser.mly

%.cmo : %.ml
ocamlc -c $<

%.cmi : %.mli
ocamlc -c $<

.PHONY: clean
clean :
rm -f parser.ml parser.mli scanner.ml *.cmo *.cmi csslang

interpret.cmo: ast.cmi
interpret.cmx: ast.cmi
csslang.cmo: scanner.cmo parser.cmi interpret.cmo
```

```
csslang.cmx: scanner.cmx parser.cmx interpret.cmx
parser.cmo: ast.cmi parser.cmi
parser.cmx: ast.cmi parser.cmi
scanner.cmo: parser.cmi
scanner.cmx: parser.cmx
parser.cmi: ast.cmi
```