

Apuntes de la Asignatura Procesadores de Lenguajes

Casiano R. León ¹

27 de febrero de 2014

¹DEIOC Universidad de La Laguna

Índice general

I PRIMERA PARTE: APUNTES DE PROCESADORES DE LENGUAJES	8
1. Expresiones Regulares y Análisis Léxico en JavaScript	9
1.1. Mozilla Developer Network: Documentación	9
1.2. Práctica: Conversor de Temperaturas	9
1.3. Práctica: Comma Separated Values. CSV	13
1.4. Comentarios y Consejos	33
1.5. Ejercicios	34
1.6. Práctica: Palabras Repetidas	36
1.7. Ejercicios	40
1.8. Ejercicios	40
1.9. Práctica: Ficheros INI	41
1.10. Práctica: Analizador Léxico para Un Subconjunto de JavaScript	49
2. Análisis Sintáctico Mediante Precedencia de Operadores en JavaScript	51
2.1. Ejemplo Simple de Intérprete: Una Calculadora	51
2.2. Análisis Top Down Usando Precedencia de Operadores	51
2.2.1. Gramática de JavaScript	51
3. Análisis Descendente mediante Parsing Expression Grammars en JavaScript	52
3.1. Conceptos Básicos para el Análisis Sintáctico	52
3.1.1. Ejercicio	53
3.2. Análisis Sintáctico Predictivo Recursivo	53
3.2.1. Introducción	53
3.2.2. Ejercicio: Recorrido del árbol en un ADPR	56
3.2.3. Ejercicio: Factores Comunes	56
3.2.4. Derivaciones a vacío	56
3.2.5. Construcción de los conjuntos de Primeros y Siguientes	57
3.2.6. Ejercicio: Construir los <i>FIRST</i>	58
3.2.7. Ejercicio: Calcular los <i>FOLLOW</i>	58
3.2.8. Práctica: Construcción de los FIRST y los FOLLOW	58
3.2.9. Gramáticas LL(1)	59
3.2.10. Ejercicio: Caracterización de una gramática LL(1)	59
3.2.11. Ejercicio: Ambigüedad y LL(1)	60
3.2.12. Práctica: Un analizador APDR	60
3.2.13. Práctica: Generación Automática de Analizadores Predictivos	60
3.3. Esquemas de Traducción	66
3.4. Recursión por la Izquierda	67
3.5. Introducción	67
3.5.1. Syntax	67
3.5.2. Semantics	68
3.5.3. Implementing parsers from parsing expression grammars	68
3.5.4. Lexical Analysis	69
3.5.5. Left recursion	69

3.5.6. Referencias y Documentación	69
3.6. Un Ejemplo Sencillo	69
3.6.1. Asociación Incorrecta para la Resta y la División	71
3.7. PegJS en los Browser	71
3.8. Eliminación de la Recursividad por la Izquierda	74
3.8.1. Eliminación de la Recursividad por la izquierda Respetando el Orden de Ejecución de las Accion	
3.8.2. Eliminando la Recursividad por la Izquierda en la Calculadora	76
3.9. Eliminación de la Recursividad por la Izquierda y Atributos Heredados	77
3.9.1. Eliminación de la Recursión por la Izquierda en la Gramática	77
3.9.2. Eliminación de la Recursión por la Izquierda en un Esquema de Traducción . .	77
3.9.3. Eliminación de la Recursividad por la Izquierda en PEGJS	78
3.10. Asociando un else con su if mas cercano	79
3.11. Comentarios Anidados	79
3.12. Un Lenguaje Dependiente del Contexto	80
3.13. Usando Pegjs con CoffeeScript	81
3.14. Práctica: Ambigüedad en C++	82
3.15. Práctica: Inventando un Lenguaje: Tortoise	84
4. Análisis Sintáctico Ascendente en JavaScript	85
4.1. Conceptos Básicos para el Análisis Sintáctico	85
4.1.1. Ejercicio	86
4.2. Ejemplo Simple en Jison	86
4.2.1. Véase También	89
4.2.2. Práctica: Secuencia de Asignaciones Simples	89
4.3. Ejemplo en Jison: Calculadora Simple	89
4.3.1. Práctica: Calculadora con Listas de Expresiones y Variables	94
4.4. Conceptos Básicos del Análisis LR	94
4.5. Construcción de las Tablas para el Análisis SLR	97
4.5.1. Los conjuntos de Primeros y Sigüientes	97
4.5.2. Construcción de las Tablas	98
4.6. Práctica: Traducción de Infijo a Postfijo	104
4.7. Práctica: Calculadora con Funciones	105
4.8. Práctica: Calculadora con Análisis de Ámbito	106
4.9. Algoritmo de Análisis LR	110
4.10. El módulo Generado por jison	111
4.10.1. Version	111
4.10.2. Gramática Inicial	111
4.10.3. Tablas	111
4.10.4. Acciones Semánticas	113
4.10.5. Tabla de Acciones y GOTOS	114
4.10.6. defaultActions	114
4.10.7. Reducciones	115
4.10.8. Desplazamientos/Shifts	116
4.10.9. Manejo de Errores	117
4.10.10. Analizador Léxico	118
4.10.11. Exportación	120
4.11. Precedencia y Asociatividad	122
4.12. Esquemas de Traducción	127
4.13. Manejo en jison de Atributos Heredados	128
4.14. Definición Dirigida por la Sintaxis	131
4.15. Ejercicios: Casos de Estudio	133
4.15.1. Un mal diseño	133
4.15.2. Gramática no LR(1)	133

4.15.3. Un Lenguaje Intrínsecamente Ambiguo	134
4.15.4. Conflicto reduce-reduce	134
4.16. Recuperación de Errores	135
4.17. Depuración en <code>jison</code>	136
4.18. Construcción del Árbol Sintáctico	136
4.19. Consejos a seguir al escribir un programa <code>jison</code>	136
5. Análisis Sintáctico Ascendente en Ruby	137
5.1. La Calculadora	137
5.1.1. Uso desde Línea de Comandos	137
5.1.2. Análisis Léxico con <code>rexical</code>	138
5.1.3. Análisis Sintáctico	138
5.2. Véase También	140
II PARTE: CREATE YOUR OWN PROGRAMMING LANGUAGE	141
6. JavaScript Review	143
6.1. Closures	143
7. Your First Compiler	144
8. Parsing	145
9. Scheem Interpreter	146
9.1. Scheem Interpreter	146
9.2. Variables	146
9.3. Setting Values	146
9.4. Putting Things Together	146
9.4.1. Unit Testing: Mocha	146
9.4.2. Karma	149
9.4.3. Grunt	155
9.4.4. GitHub Project Pages	159
10.Functions and all that	161
11. Inventing a language for turtle graphics	162
III HERRAMIENTAS	163
12.Heroku	164
12.1. Introducción	164
12.2. Logging	172
12.3. Troubleshooting	174
12.4. Configuration	176
12.5. Make Heroku run non-master Git branch	176
12.6. Account Verification and add-ons	177
12.7. Véase	177
IV PARTE: BITÁCORA DEL CURSO	178
13.2014	179
13.1. 01	179
13.1.1. Semana del 27/01/14 al 01/02/2014	179

13.2. 02	179
13.2.1. Semana del 4/02/14 al 7/02/2014	179
13.2.2. Semana del 24/02/14 al 02/03/14. Repaso para el micro-examen del 05/03/14 .	179
13.3. Proyecto: Diseña e Implementa un Lenguaje de Dominio Específico	189

Índice de figuras

1.1. Ejemplo de pantalla de La aplicación para el Análisis de Datos en Formato CSV . . .	14
3.1. pegjs en la web	74
4.1. NFA que reconoce los prefijos viables	97
4.2. DFA equivalente al NFA de la figura 4.1	99
4.3. DFA construido por Jison	112

Índice de cuadros

A Juana

*For it is in teaching that we learn
And it is in understanding that we are understood*

Agradecimientos/Acknowledgments

I'd like to thank

*A mis alumnos de Procesadores de Lenguajes del Grado de Informática de la Escuela
Superior de Informática en la Universidad de La Laguna*

Parte I

PRIMERA PARTE: APUNTES DE PROCESADORES DE LENGUAJES

Capítulo 1

Expresiones Regulares y Análisis Léxico en JavaScript

1.1. Mozilla Developer Network: Documentación

1. RegExp Objects
2. exec
3. search
4. match
5. replace

1.2. Práctica: Conversor de Temperaturas

Véase <https://bitbucket.org/casiano/pl-grado-temperature-converter/src>.

```
[~/srcPLgrado/temperature(master)]$ pwd -P
/Users/casiano/local/src/javascript/PLgrado/temperature # 27/01/2014
```

index.html

```
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>JavaScript Temperature Converter</title>
    <link href="global.css" rel="stylesheet" type="text/css">

    <script type="text/javascript" src="temperature.js"></script>
  </head>
  <body>
    <h1>Temperature Converter</h1>
    <table>
      <tr>
        <th>Enter Temperature (examples: 32F, 45C, -2.5f):</th>
        <td><input id="original" onchange="calculate();"></td>
      </tr>
      <tr>
        <th>Converted Temperature:</th>

```

```

        <td><span class="output" id="converted"></span></td>
    </tr>
</table>
</body>
</html>

```

global.css

```

th, td      { vertical-align: top; text-align: right; font-size: large; }
#converted  { color: red; font-weight: bold; font-size: large;        }
input       { text-align: right; border: none; font-size: large;      }
body
{
    background-color: #b0c4de; /* blue */
    font-size: large;
}

```

temperature.js

```

"use strict"; // Use ECMAScript 5 strict mode in browsers that support it
function calculate() {
    var result;
    var original      = document.getElementById(".....");
    var temp = original.value;
    var regexp = /...../;

    var m = temp.match(.....);

    if (m) {
        var num = ....;
        var type = ....;
        num = parseFloat(num);
        if (type == 'c' || type == 'C') {
            result = (num * 9/5)+32;
            result = .....
        }
        else {
            result = (num - 32)*5/9;
            result = .....
        }
        converted.innerHTML = result;
    }
    else {
        converted.innerHTML = "ERROR! Try something like '-4.2C' instead";
    }
}

```

Despliegue

- Deberá desplegar la aplicación en GitHub Pages como página de proyecto. Vea la sección *GitHub Project Pages* 9.4.4.

Mocha y Chai Mocha is a test framework while Chai is an expectation one.

Let's say Mocha setups and describes test suites and Chai provides convenient helpers to perform all kinds of assertions against your JavaScript code.

Pruebas: Estructura

Instale mocha.

```
$ npm install -g mocha
```

Creemos la estructura para las pruebas:

```
$ mocha init tests
```

```
$ tree tests
```

```
tests
|-- index.html
|-- mocha.css
|-- mocha.js
'-- tests.js
```

Añadimos `chai.js` (Véase <http://chaijs.com/guide/installation/>) al directorio `tests`.

The latest tagged version will be available for hot-linking at <http://chaijs.com/chai.js>.

If you prefer to host yourself, use the `chai.js` file from the root of the github project.

```
[~/srcPLgrado/temperature(master)]$ tree tests/
tests/
|-- chai.js
|-- index.html
|-- mocha.css
|-- mocha.js
'-- tests.js
```

0 directories, 5 files

Podemos encontrar un ejemplo de unit testing en JavaScript en el browser con el testing framework Mocha y Chai en el repositorio <https://github.com/ludovicofischer/mocha-chai-browser-demo>.

Pruebas: index.html

Modificamos `index.html` para

- Cargar `chai.js`
- Cargar `temperature.js`
- Usar el estilo `mocha.setup('tdd')`:
- Imitar la página `index.html` con los correspondientes `input` y `span`:

```
<input id="original" placeholder="32F" size="50">
<span class="output" id="converted"></span>
```

```
[~/srcPLgrado/temperature(master)]$ cat tests/index.html
<!DOCTYPE html>
<html>
  <head>
    <title>Mocha</title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="mocha.css" />
  </head>
  <body>
    <div id="mocha"></div>
    <input id="original" placeholder="32F" size="50">
    <span class="output" id="converted"></span>

    <script src="chai.js"></script>
    <script src="mocha.js"></script>
    <script src="../temperature.js"></script>
    <script>mocha.setup('tdd')</script>
    <script src="tests.js"></script>

    <script>
      mocha.run();
    </script>
  </body>
</html>
```

Pruebas: Añadir los tests

The "TDDinterface provides

- suite()
- test()
- setup()
- teardown().

```
[~/srcPLgrado/temperature(master)]$ cat tests/tests.js
var assert = chai.assert;

suite('temperature', function() {
  test('32F = 0C', function() {
    original.value = "32F";
    calculate();
    assert.deepEqual(converted.innerHTML, "0.0 Celsius");
  });
  test('45C = 113.0 Farenheit', function() {
    original.value = "45C";
    calculate();
    assert.deepEqual(converted.innerHTML, "113.0 Farenheit");
  });
  test('5X = error', function() {
    original.value = "5X";
    calculate();
    assert.match(converted.innerHTML, /ERROR/);
  });
});
```

```
});  
});
```

Pruebas: Véase

- Testing your frontend JavaScript code using mocha, chai, and sinon by Nicolas Perriault

1.3. Práctica: Comma Separated Values. CSV

Donde

```
[~/srcPLgrado/csv(master)]$ pwd -P  
/Users/casiano/local/src/javascript/PLgrado/csv
```

Véase <https://bitbucket.org/casiano/pl-grado-csv/src> y <https://github.com/crguezl/csv>.

Introducción al formato CSV

Véase Comma Separated Values en la Wikipedia:

A comma-separated values (CSV) file stores tabular data (numbers and text) in plain-text form. A CSV file consists of any number of records, separated by line breaks of some kind; each record consists of fields, separated by a comma. All records have an identical sequence of fields.

Ejemplo de ejecución

Véase la página en <http://crguezl.github.io/csv/>. Pruebe a dar como entrada cualquiera de estas dos

```
[~/srcPLgrado/csv(gh-pages)]$ cat input.txt  
"producto",          "precio"  
"camisa",            "4,3"  
"libro de O\"Reilly", "7,2"
```

```
[~/srcPLgrado/csv(gh-pages)]$ cat input2.txt  
"producto",          "precio" "fecha"  
"camisa",            "4,3",    "14/01"  
"libro de O\"Reilly", "7,2"    "13/02"
```

Pruebe también a dar alguna entrada errónea.

Aproximación al análisis mediante expresiones regulares de CSV Una primera aproximación sería hacer `split` por las comas:

```
> x = '"earth",1,"moon",9.374'  
'"earth",1,"moon",9.374'  
> y = x.split(/,/)  
[ '"earth"', '1', '"moon"', '9.374' ]
```

Esta solución deja las comillas dobles en los campos entrecomillados. Peor aún, los campos entrecomillados pueden contener comas, en cuyo caso la división proporcionada por `split` sería errónea:

```
> x = '"earth, mars",1,"moon, fobos",9.374'  
'"earth, mars",1,"moon, fobos",9.374'  
> y = x.split(/,/)  
[ '"earth', ' mars"', '1', '"moon', ' fobos"', '9.374' ]
```

La siguiente expresión regular reconoce cadenas de comillas dobles con secuencias de escape seguidas opcionalmente de una coma:



Figura 1.1: Ejemplo de pantalla de La aplicación para el Análisis de Datos en Formato CSV

```
> x = '"earth, mars",1,"moon, fobos",9.374'
'"earth, mars",1,"moon, fobos",9.374'
> r = /"((?:[^\\"|\\\.]*)"\s*,?/g
/((?:[^\\"|\\\.]*)"\s*,?/g
> w = x.match(r)
[ '"earth, mars",', '"moon, fobos",' ]
```

If your regular expression uses the `g` flag, you can use the `exec` or `match` methods multiple times to find successive matches in the same string. When you do so, the search starts at the substring of string specified by the regular expression's `lastIndex` property.

Javascript sub-matches stop working when the `g` modifier is set:

```
> text = 'test test test test'
'test test test test'
> text.match(/t(e)(s)t/)
[ 'test', 'e', 's', index: 0, input: 'test test test test' ]
> text.match(/t(e)(s)t/g)
[ 'test', 'test', 'test', 'test' ]
```

Sin embargo el método `exec` de las expresiones regulares si que conserva las subexpresiones que casan con los paréntesis:

```
> r = /t(e)(s)t/g
/t(e)(s)t/g
> text = 'test test test test'
'test test test test'
> while (m = r.exec(text)) {
... console.log(m);
... }
[ 'test', 'e', 's', index: 0, input: 'test test test test' ]
[ 'test', 'e', 's', index: 5, input: 'test test test test' ]
```



```
[ 'test', 'e', 's', index: 10, input: 'test test test test' ]
[ 'test', 'e', 's', index: 15, input: 'test test test test' ]
undefined
```

Another catch to remember is that `exec()` doesn't return the matches in one big array: it keeps returning matches until it runs out, in which case it returns `null`.

Véase

- Javascript Regex and Submatches en StackOverflow.
- La sección *Ejercicios* 1.5

Esta otra expresión regular `/([^\,]+)?|\s*,/` actúa de forma parecida al `split`. Reconoce secuencias no vacías de caracteres que no contienen comas seguidas opcionalmente de una coma o bien una sola coma (precedida opcionalmente de blancos):

```
> x = '"earth, mars",1,"moon, fobos",9.374'
'"earth, mars",1,"moon, fobos",9.374'
> r = /([^\,]+)?|\s*/,/g
/([^\,]+)?|\s*/,/g
> w = x.match(r)
[ '"earth,', ' mars', '1', '"moon,', ' fobos', '9.374' ]
```

La siguiente expresión regular es la unión de dos:

- Cadenas de dobles comillas seguidas de una coma opcional entre espacios en blanco
- Cadenas que no tienen comas

```
> x = '"earth, mars",1,"moon, fobos",9.374'
'"earth, mars",1,"moon, fobos",9.374'
> r = /\s*"((?:[^\,\\]|\\.)*"\s*,?\s*"([^\,]+)?|\s*/,/g
/\s*"((?:[^\,\\]|\\.)*"\s*,?\s*"([^\,]+)?|\s*/,/g
> w = x.match(r)
[ '"earth, mars', '1', '"moon, fobos', '9.374' ]
```

El operador `|` trabaja en circuito corto:

```
> r = /(ba?)|(b)/
/(ba?)|(b)/
> r.exec("ba")
[ 'ba', 'ba', undefined, index: 0, input: 'ba' ]
> r = /(b)|(ba?)/
/(b)|(ba?)/
> r.exec("ba")
[ 'b', 'b', undefined, index: 0, input: 'ba' ]
```

Si usamos `exec` tenemos:

```
> x = '"earth, mars",1,"moon, fobos",9.374'
'"earth, mars",1,"moon, fobos",9.374'
> r = /\s*"((?:[^\,\\]|\\.)*"\s*,?\s*"([^\,]+)?|\s*/,/g
/\s*"((?:[^\,\\]|\\.)*"\s*,?\s*"([^\,]+)?|\s*/,/g
> while (m = r.exec(x)) { console.log(m); }
[ '"earth, mars', 'earth, mars', undefined, index: 0,
  input: '"earth, mars",1,"moon, fobos",9.374' ]
[ '1', undefined, '1', index: 14,
  input: '"earth, mars",1,"moon, fobos",9.374' ]
```

```
[ 'moon, fobos',, 'moon, fobos', undefined, index: 16,
  input: '"earth, mars",1,"moon, fobos",9.374' ]
[ '9.374', undefined, '9.374', index: 30,
  input: '"earth, mars",1,"moon, fobos",9.374' ]
undefined
```

1. RegExp Objects

The `RegExp` constructor creates a regular expression object for matching text with a pattern.

Literal and constructor notations are possible:

```
/pattern/flags;
new RegExp(pattern [, flags]);
```

- The literal notation provides compilation of the regular expression when the expression is evaluated.
- Use literal notation when the regular expression will remain constant.
- For example, if you use literal notation to construct a regular expression used in a loop, the regular expression won't be recompiled on each iteration.
- The constructor of the regular expression object, for example, `new RegExp("ab+c")`, provides runtime compilation of the regular expression.
- Use the constructor function when you know the regular expression pattern will be changing, or you don't know the pattern and are getting it from another source, such as user input.
- When using the constructor function, the normal string escape rules (preceding special characters with `\` when included in a string) are necessary. For example, the following are equivalent:

```
var re = /\w+/;
var re = new RegExp("\\w+");
```

2. exec

3. search

```
str.search(regex)
```

If successful, `search` returns the index of the regular expression inside the string. Otherwise, it returns `-1`.

When you want to know whether a pattern is found in a string use `search` (similar to the regular expression `test` method); for more information (but slower execution) use `match` (similar to the regular expression `exec` method).

4. match

5. replace

The `replace()` method returns a new string with some or all matches of a pattern replaced by a replacement. The pattern can be a string or a `RegExp`, and the replacement can be a string or a function to be called for each match.

```
> re = /apples/gi
/apples/gi
> str = "Apples are round, and apples are juicy."
'Apples are round, and apples are juicy.'
> newstr = str.replace(re, "oranges")
'oranges are round, and oranges are juicy.'
```

The replacement string can be a function to be invoked to create the new substring (to put in place of the substring received from parameter #1). The arguments supplied to this function are:

Possible name	Supplied value
match	The matched substring. (Corresponds to \$&.)
p1, p2, ...	The nth parenthesized submatch string, provided the first argument to replace was a regular expression. (Corresponds to \$1, \$2, etc.) For example, if /(\a+)(\b+)/, was given, p1 would be \a+ and p2 for \b+.
offset	The offset of the matched substring within the total string being examined. For example, if the string was abcd, and the matched substring was bc, then this argument would be 1.
string	The total string being examined

```
[~/javascript/learning]$ pwd -P
/Users/casiano/local/src/javascript/learning
[~/javascript/learning]$ cat f2c.js
#!/usr/bin/env node
function f2c(x)
{
    function convert(str, p1, offset, s)
    {
        return ((p1-32) * 5/9) + "C";
    }
    var s = String(x);
    var test = /(\d+(?:\.\d*)?)F\b/g;
    return s.replace(test, convert);
}

var arg = process.argv[2] || "32F";
console.log(f2c(arg));

[~/javascript/learning]$ ./f2c.js 100F
37.77777777777778C
[~/javascript/learning]$ ./f2c.js
0C
```

index.html

```
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>CSV Analyzer</title>
    <link href="global.css" rel="stylesheet" type="text/css">

    <script type="text/javascript" src="../../underscore/underscore.js"></script>
    <script type="text/javascript" src="../../jquery/starterkit/jquery.js"></script>
    <script type="text/javascript" src="csv.js"></script>
  </head>
  <body>
    <h1>Comma Separated Value Analyzer</h1>
    <div>
      <i>Write a CSV string. Click the table button. The program outputs a table with the spec</i>
    </div>
    <table>
      <tr>
```

```

        <th>CSV string:</th> <!-- autofocus attribute is HTML5 -->
        <td><textarea autofocus cols = "80" rows = "5" id="original"></textarea></td>
    </tr>
</table>
<button type="button">table:</button><br>
<span class="output" id="finaltable"></span>
</body>
</html>

```

jQuery

jQuery (Descarga la librería)

jQuery is a cross-platform JavaScript library designed to simplify the client-side scripting of HTML.

- It was released in January 2006 at BarCamp NYC by John Resig.
- It is currently developed by a team of developers led by Dave Methvin.
- jQuery is the most popular JavaScript library in use today
- jQuery's syntax is designed to make it easier to navigate a document, select DOM elements, create animations, handle events, and develop Ajax applications.
- The set of jQuery core features — DOM element selections, traversal and manipulation — enabled by its selector engine (named "Sizzle" from v1.3), created a new "programming style", fusing algorithms and DOM-data-structures; and influenced the architecture of other JavaScript frameworks like YUI v3 and Dojo.
-

How JQuery Works

- Véase How jQuery Works
- <https://github.com/crguezl/how-jquery-works-tutorial> en GitHub
- ```
[~/javascript/jquery(master)]$ pwd -P
/Users/casiano/local/src/javascript/jquery
```

```

~/javascript/jquery(master)]$ cat index.html
<!doctype html>
<html>
<head>
 <meta charset="utf-8" />
 <title>Demo</title>
</head>
<body>
 jQuery
 <script src="starterkit/jquery.js"></script>
 <script>

 // Your code goes here.

 </script>
</body>
</html>

```

To ensure that their code runs after the browser finishes loading the document, many JavaScript programmers wrap their code in an onload function:

```
window.onload = function() { alert("welcome"); }
```

Unfortunately, the code doesn't run until all images are finished downloading, including banner ads. To run code as soon as the document is ready to be manipulated, jQuery has a statement known as the ready event:

```
$(document).ready(function() {
 // Your code here.
});
```

For click and most other events, you can prevent the default behavior by calling `event.preventDefault()` in the event handler. If this method is called, the default action of the event will not be triggered. For example, clicked anchors will not take the browser to a new URL.

```
[~/javascript/jquery(master)]$ cat index2.html
<!doctype html>
<html>
<head>
 <meta charset="utf-8" />
 <title>Demo</title>
</head>
<body>
 jQuery
 <script src="starterkit/jquery.js"></script>
 <script>

 $(document).ready(function() {
 $("a").click(function(event) {
 alert("The link will no longer take you to jquery.com");
 event.preventDefault();
 });
 });

 </script>
</body>
</html>
```

Borrowing from CSS 1–3, and then adding its own, jQuery offers a powerful set of tools for matching a set of elements in a document.

See jQuery Category: Selectors.

Another common task is adding or removing a class. jQuery also provides some handy effects.

```
[~/javascript/jquery(master)]$ cat index3.html
<!doctype html>
<html>
<head>
 <meta charset="utf-8" />
 <style>
 a.test { font-weight: bold; }
 </style>
 <title>Demo</title>
</head>
```

```

<body>
 jQuery
 <script src="starterkit/jquery.js"></script>
 <script>

 $(document).ready(function() {
 $("a").click(function(event) {
 $("a").addClass("test");
 alert("The link will no longer take you to jquery.com");
 event.preventDefault();
 $("a").removeClass("test");
 $(this).hide("slow");
 $(this).show("slow");
 });
 });

 </script>
</body>
</html>

```

- In JavaScript **this** always refers to the *owner* of the function we're executing, or rather, *to the object that a function is a method of*.
- When we define our function `tutu()` in a page, its owner is the page, or rather, the **window** object (or **global** object) of JavaScript.
- An **onclick** property, though, is owned by the HTML element it belongs to.
- The method `.addClass( className )` adds the specified class(es) to each of the set of matched elements.

`className` is a String containing one or more space-separated classes to be added to the class attribute of each matched element.

This method does not replace a class. It simply adds the class, appending it to any which may already be assigned to the elements.

- The method `.removeClass( [className] )` removes a single class, multiple classes, or all classes from each element in the set of matched elements.

If a class name is included as a parameter, then only that class will be removed from the set of matched elements. If no class names are specified in the parameter, all classes will be removed.

This method is often used with `.addClass()` to switch elements' classes from one to another, like so:

```
$("p").removeClass("myClass noClass").addClass("yourClass");
```

JavaScript enables you to freely pass functions around to be executed at a later time. A *callback* is a function that is passed as an argument to another function and is usually executed after its parent function has completed.

Callbacks are special because they wait to execute until their parent finishes or some event occurs. Meanwhile, the browser can be executing other functions or doing all sorts of other work.

```

[~/javascript/jquery(master)]$ cat app.rb
require 'sinatra'

set :public_folder, File.dirname(__FILE__) + '/starterkit'

```

```

get '/' do
 erb :index
end

```

```

get '/chuchu' do
 if request.xhr?
 "hello world!"
 else
 erb :tutu
 end
end

```

```

__END__

```

```

@@layout
<!DOCTYPE html>
<html>
 <head>
 <meta charset="utf-8" />
 <title>Demo</title>
 </head>
 <body>
 jQuery
 <div class="result"></div>
 <script src="jquery.js"></script>
 <%= yield %>
 </body>
</html>

```

```

@@index
<script>
$(document).ready(function() {
 $("a").click(function(event) {
 event.preventDefault();
 $.get("/chuchu", function(data) {
 $(".result").html(data);
 alert("Load was performed.");
 });
 });
});
</script>

```

```

@@tutu
<h1>Not an Ajax Request!</h1>

```

- `jQuery.get( url [, data ] [, success(data, textStatus, jqXHR) ] [, dataType ] )` load data from the server using a HTTP GET request.
- `url`  
Type: String  
A string containing the URL to which the request is sent.
- `data`

Type: PlainObject or String

A plain object or string that is sent to the server with the request.

- **success(data, textStatus, jqXHR)**

Type: Function()

A callback function that is executed if the request succeeds.

- **dataType**

Type: String

The type of data expected from the server. Default: Intelligent Guess (xml, json, script, or html).

To use callbacks, it is important to know how to pass them into their parent function.

Executing callbacks with arguments can be tricky.

This code example will not work:

```
$.get("myhtmlpage.html", myCallBack(param1, param2));
```

The reason this fails is that the code executes

```
myCallBack(param1, param2)
```

immediately and then passes `myCallBack()`'s return value as the second parameter to `$.get()`.

We actually want to pass the function `myCallBack`, not `myCallBack( param1, param2 )`'s return value (which might or might not be a function).

So, how to pass in `myCallBack()` and include arguments?

To defer executing `myCallBack()` with its parameters, you can use an anonymous function as a wrapper.

```
[~/javascript/jquery(master)]$ cat app2.rb
require 'sinatra'

set :public_folder, File.dirname(__FILE__) + '/starterkit'

get '/' do
 erb :index
end

get '/chuchu' do
 if request.xhr? # is an ajax request
 "hello world!"
 else
 erb :tutu
 end
end

__END__

@@layout
<!DOCTYPE html>
<html>
 <head>
 <meta charset="utf-8" />
 <title>Demo</title>
```



```

</head>
<body>
 jQuery
 <div class="result"></div>
 <script src="jquery.js"></script>
 <%= yield %>
</body>
</html>

```

```

@@tutu
 <h1>Not an Ajax Request!</h1>

```

```

@@index
 <script>
 var param = "chuchu param";
 var handler = function(data, textStatus, jqXHR, param) {
 $(".result").html(data);
 alert("Load was performed.\n"+
 "$data = "+data+
 "\ntextStatus = "+textStatus+
 "\njqXHR = "+JSON.stringify(jqXHR)+
 "\nparam = "+param);
 };
 $(document).ready(function() {
 $("a").click(function(event) {
 event.preventDefault();
 $.get("/chuchu", function(data, textStatus, jqXHR) {
 handler(data, textStatus, jqXHR, param);
 });
 });
 });
 </script>

```

El ejemplo en `app2.rb` puede verse desplegado en Heroku: <http://jquery-tutorial.herokuapp.com/>

**JSON.stringify()** The `JSON.stringify()` method converts a value to `JSON`, optionally replacing values if a `replacer` function is specified, or optionally including only the specified properties if a `replacer` array is specified.

`JSON.stringify(value[, replacer [, space]])`

- **value**

The value to convert to a JSON string.

- **replacer**

- If a function, transforms values and properties encountered while stringifying;
- if an array, specifies the set of properties included in objects in the final string.

- **space**

Causes the resulting string to be pretty-printed.

See another example of use in <http://jsfiddle.net/casiano/j7tsF/>. To learn to use JSFiddle with the YouTube video [How to use JSFiddle](#) by Jason Diamond

## Underscore

Underscore: is a utility-belt library for JavaScript that provides a lot of the functional programming support that you would expect in Ruby.

- Underscore provides functions that support methods like:  
`map`, `select`, `invoke`
- as well as more specialized helpers:  
function binding, javascript templating, deep equality testing, and so on.
- Cargando la librería:

```
[~/javascript/jquery(master)]$ node
> 2+3
5
> _
5
> uu = require('underscore')
{ [Function]
 _: [Circular],
 VERSION: '1.5.2',
 forEach: [Function],
 each: [Function],
 collect: [Function],
 map: [Function],
 inject: [Function],
 reduce: [Function],

 chain: [Function] }
```

- `each`:

```
> uu.each([1, 2, 3], function(x) { console.log(x*x); })
1
4
9
```

- `map`:

```
> uu.map([1, 2, 3], function(num){ return num * 3; })
[3, 6, 9]
```

- `invoke`

```
> z = [[6,9,1],[7,3,9]]
[[6, 9, 1], [7, 3, 9]]
> uu.invoke(z, 'sort')
[[1, 6, 9], [3, 7, 9]]
> uu.invoke(z, 'sort', function(a, b) { return b-a; })
[[9, 6, 1], [9, 7, 3]]
```

- `reduce`:

```

> uu.reduce([1, 2, 3, 4], function(s, num){ return s + num; }, 0)
10
> uu.reduce([1, 2, 3, 4], function(s, num){ return s * num; }, 1)
24
> uu.reduce([1, 2, 3, 4], function(s, num){ return Math.max(s, num); }, -1)
4
> uu.reduce([1, 2, 3, 4], function(s, num){ return Math.min(s, num); }, 99)
1

```

- **filter:** (select is an alias for filter)

```

> uu.filter([1, 2, 3, 4, 5, 6], function(num){ return num % 2 == 0; })
[2, 4, 6]

```

- **isEqual**

```

> a = {a:[1,2,3], b: { c: 1, d: [5,6]}}
{ a: [1, 2, 3],
 b: { c: 1, d: [5, 6] } }
> b = {a:[1,2,3], b: { c: 1, d: [5,6]}}
{ a: [1, 2, 3],
 b: { c: 1, d: [5, 6] } }
> a == b
false
> uu.isEqual(a,b)
true

```

- **bind**

```

> func = function(greeting){ return greeting + ': ' + this.name }
[Function]
> func = uu.bind(func, {name: 'moe'})
[Function]
> func('hello')
'hello: moe'
> func = uu.bind(func, {name: 'moe'}, 'hi')
[Function]
> func()
'hi: moe'
>

```

## Templates en Underscore

- Underscore: template

```

_.template(templateString, [data], [settings])

```

Compiles JavaScript templates into functions that can be evaluated for rendering. Useful for rendering complicated bits of HTML from a JavaScript object or from *JSON* data sources.

JSON, or *JavaScript Object Notation*, is an open standard format that uses human-readable text to transmit data objects consisting of attribute–value pairs. It is used primarily to transmit data between a server and web application, as an alternative to XML. Although originally derived from the JavaScript scripting language, JSON is a language-independent data format, and code for parsing and generating JSON data is readily available in a large variety of programming languages.

- Template functions can both interpolate variables, using `<%= ... %>`,

```
> compiled = uu.template("hello: <%= name %>")
{ [Function]
 source: 'function(obj){
 var __t,__p=\'\', __j=Array.prototype.join, i
 print=function(){__p+=__j.call(arguments,\'\');};
 with(obj||{}){
 __p+=\'hello: \' + ((__t=(name))==null?\'\':__t)+ \'\'
 }
 return __p;
 }'
}
> compiled({name: 'moe'})
'hello: moe'
```

- as well as execute arbitrary JavaScript code, with `<% ... %>`.

```
> uu = require('underscore')
> list = "\
... <% _.each(people, function(name) { %>\
..... <%= name %>\
... <% }); %>"
'<% _.each(people, function(name) { %> <%= name %> <% }); %>'
> uu.template(list, {people: ['moe', 'curly', 'larry']})
' moe curly larry '
```

- When you evaluate a template function, pass in a data object that has properties corresponding to the template's free variables.

If you're writing a one-off, like in the example above, you can pass the data object as the second parameter to template in order to render immediately instead of returning a template function.

- If you wish to interpolate a value, and have it be HTML-escaped, use `<%- ... %>`

```
> template = uu.template("<%- value %>")
{ [Function]
 source: 'function(obj){
 var __t,__p=\'\', __j=Array.prototype.join, print=function(){__p+=__j.call(argument
 with(obj||{}){
 __p+=\'\'+
 ((__t=(value))==null?\'\':_.escape(__t))+
 \'\';
 }
 return __p;
 }'
}
> template({value: '<script>'})
'<script>'
```

- The `settings` argument should be a hash containing any `_.templateSettings` that should be overridden.

```
_.template("Using 'with': <%= data.answer %>", {answer: 'no'}, {variable: 'data'});
=> "Using 'with': no"
```

Another example:

```

template = uu.template("{{ value }}",{value: 4 },
 { interpolate: /\{\{(.+?)\}\}/g })
'4'

```

You can also use `print` from within JavaScript code. This is sometimes more convenient than using `<%= ... %>`.

```

> compiled = uu.template("<% print('Hello ' + epithet); %>")
{ [Function]
 source: 'function(obj){\n
 var __t,__p=\'\',
 __j=Array.prototype.join,print=function(){
 __p+=__j.call(arguments,\'\');};\n
 with(obj||{}){\n
 __p+=\'\';\n print(\'Hello \' + epithet); \n
 __p+=\'\';\n}\n
 return __p;\n
 }'
}
> compiled({ epithet : 'stooge' })
'Hello stooge'
>

```

If ERB-style delimiters aren't your cup of tea, you can change Underscore's template settings to use different symbols to set off interpolated code:

- Define an `interpolate` regex to match expressions that should be interpolated verbatim,
- an `escape` regex to match expressions that should be inserted after being HTML escaped, and
- an `evaluate` regex to match expressions that should be evaluated without insertion into the resulting string.
- You may define or omit any combination of the three.
- For example, to perform `Mustache.js` style templating:

```

_.templateSettings = {
 interpolate: /\{\{(.+?)\}\}/g
};

var template = _.template("Hello {{ name }}!");
template({name: "Mustache"});
=> "Hello Mustache!"

```

- `escape`:

```

> uu.templateSettings.escape = /\{\{-(.*)\}\}/g
/\{\{-(.*)\}\}/g
> compiled = uu.template("Escaped: {{- value }}\nNot escaped: {{ value }}")
{ [Function]
 source: 'function(obj){\nvar __t,__p=\'\',__j=Array.prototype.join,print=function()
> compiled({value: 'Hello, world!'})
'Escaped: Hello, world!\nNot escaped: {{ value }}'

```

- Another example:

```

> uu.templateSettings = {
..... interpolate: /\<\@=(.+)\>/gim,
..... evaluate: /\<\@(.+)\>/gim
..... }
{ interpolate: /\<\@=(.+)\>/gim,
 evaluate: /\<\@(.+)\>/gim }
> s = " <@ _.each([0,1,2,3,4], function(i) { @> <p><@= i @></p> <@ }>); @>"
' <@ _.each([0,1,2,3,4], function(i) { @> <p><@= i @></p> <@ }>); @>'
> uu.template(s,{})
' <p>0</p> <p>1</p> <p>2</p> <p>3</p> <p>4</p> '

```

By default, template places the values from your data in the local scope via the **with** statement. The **with** statement adds the given object to the head of this scope chain during the evaluation of its statement body:

```

> with (Math) {
... s = PI*2;
... }
6.283185307179586
> z = { x : 1, y : 2 }
{ x: 1, y: 2 }
> with (z) {
... console.log(y);
... }
2
undefined

```

However, you can specify a single variable name with the variable **setting**. This improves the speed at which a template is able to render.

```

_.template("Using 'with': <%= data.answer %>", {answer: 'no'}, {variable: 'data'});
=> "Using 'with': no"

```

- JSFIDDLE: underscore templates
- Stackoverflow::how to use Underscore template

## Content delivery network or content distribution network (CDN)

Una CDN que provee **underscore** esta en <http://cdnjs.com/>:

```

<script type="text/javascript" src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.5.2">
<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.10.2/jquery.min.js"></script>

```

A content delivery network or content distribution network (CDN) is a large distributed system of servers deployed in multiple data centers across the Internet. The goal of a CDN is to serve content to end-users with high availability and high performance. CDNs serve a large fraction of the Internet content today, including

- web objects (text, graphics and scripts),
- downloadable objects (media files, software, documents), applications (e-commerce, portals),
- live streaming media, on-demand streaming media, and social networks.

Google provee también un servicio CDN para los desarrolladores en <https://developers.google.com/speed/librar>

## textarea, autofocus y button

### 1. textarea:

The `<textarea>` tag defines a multi-line text input control.

A *text area* can hold an unlimited number of characters, and the text renders in a fixed-width font (usually Courier).

The size of a text area can be specified by the `cols` and `rows` attributes, or through CSS' `height` and `width` properties.

`cols` and `rows` consider the font size. `height` and `width` aren't.

### 2. autofocus.

The `autofocus` attribute is a boolean attribute.

When present, it specifies that the text area should automatically get focus when the page loads.

Véase también [1]

### 3. button:

The `<button>` tag defines a clickable button.

Inside a `<button>` element you can put content, like text or images.

## Local Storage (HTML5 Web Storage)

*Web storage* and *DOM storage* (document object model) are web application software methods and protocols used for storing data in a web browser.

- Web storage supports persistent data storage, similar to cookies but with a greatly enhanced capacity and no information stored in the HTTP request header.
- Local Storage nos permite almacenar hasta 5MB del lado del cliente por dominio, esto nos permite ahora hacer aplicaciones mas robustas y con mas posibilidades. Las Cookies ofrecen algo parecido, pero con el limite de 100kb.
- There are two main web storage types: *local storage* and *session storage*, behaving similarly to persistent cookies and session cookies respectively.
- Unlike cookies, which can be accessed by both the server and client side, web storage falls exclusively under the purview of client-side scripting
- The HTML5 localStorage object is isolated per domain (the same segregation rules as the same origin policy).

The same-origin policy permits scripts running on pages originating from the same site – a combination of scheme, hostname, and port number – to access each other's DOM with no specific restrictions, but prevents access to DOM on different sites.

Véase:

- Ejemplo en GitHub: <https://github.com/crguezl/web-storage-example>

```
[~/javascript/local_storage(master)]$ pwd -P
/Users/casiano/local/src/javascript/local_storage
```

- Como usar localStorage
- HTML5 Web Storage
- W3C Web Storage

- Using HTML5 localStorage To Store JSON Options for persistent storage of complex JavaScript objects in HTML5 by Dan Cruickshank
- HTML5 Cookbook. Christopher Schmitt, Kyle Simpson .O'Reilly Media, Inc.”, Nov 7, 2011 Chapter 10. Section 2: LocalStorage

While Chrome does not provide a UI for clearing localStorage, there is an API that will either clear a specific key or the entire localStorage object on a website.

```
//Clears the value of MyKey
window.localStorage.clear("MyKey");
```

```
//Clears all the local storage data
window.localStorage.clear();
```

Once done, localStorage will be cleared. Note that this affects all web pages on a single domain, so if you clear localStorage for [jsfiddle.net/index.html](http://jsfiddle.net/index.html) (assuming that's the page you're on), then it clears it for all other pages on that site.

## global.css

```
html *
{
 font-size: large;
 /* The !important ensures that nothing can override what you've set in this style (unless i
 font-family: Arial;
}

h1 { text-align: center; font-size: x-large; }
th, td { vertical-align: top; text-align: right; }
/* #finaltable * { color: white; background-color: black; } */

/* #finaltable table { border-collapse: collapse; } */
/* #finaltable table, td { border: 1px solid white; } */
#finaltable: hover td { background-color: blue; }
tr:nth-child(odd) { background-color: #eee; }
tr:nth-child(even) { background-color: #00FF66; }
input { text-align: right; border: none; } /* Align input to the right */
textarea { border: outset; border-color: white; }
table { border: inset; border-color: white; }
table.center { margin-left: auto; margin-right: auto; }
#result { border-color: red; }
tr.error { background-color: red; }
body
{
 background-color: #b0c4de; /* blue */
}
```

### 1. Introducción a las pseudo clases de CSS3

Una pseudo clase es un estado o uso predefinido de un elemento al que se le puede aplicar un estilo independientemente de su estado por defecto. Existen cuatro tipos diferentes de pseudo clases:

- Links: Estas pseudo clases se usan para dar estilo al enlace tanto en su estado normal por defecto como cuando ya ha sido visitado, mientras mantenemos el cursor encima de él o cuando hacemos click en él



- Dinamicas: Estas pseudo clases pueden ser aplicadas a cualquier elemento para definir como se muestran cuando el cursor está situado sobre ellos, o haciendo click en ellos o bien cuando son seleccionados
- Estructurales: Permiten dar estilo a elementos basándonos en una posición numérica exacta del elemento
- Otras: Algunos elementos pueden ser estilizados de manera diferente basándonos en el lenguaje o que tipo de etiqueta no son

## 2. CSS pattern matching

In CSS, *pattern matching* rules determine which style rules apply to elements in the document tree. These patterns, called selectors, may range from simple element names to rich contextual patterns. If all conditions in the pattern are true for a certain element, the selector matches the element.

The universal selector, written `*`, matches the name of any element type. It matches any single element in the document tree.

For example, this rule set will be applied to every element in a document:

```
* {
 margin: 0;
 padding: 0;
}
```

## 3. CSS class selectors

Working with HTML, authors may use the period (`.`) notation as an alternative to the `~=` notation when representing the class attribute. Thus, for HTML, `div.value` and `div[class~=value]` have the same meaning. The attribute value must immediately follow the *period* (`.`).

## 4. CSS3: nth-child() selector

The `:nth-child(n)` selector matches every element that is the `n`th child, regardless of type, of its parent.

`n` can be a number, a keyword, or a formula.

## 5. The CSS border properties allow you to specify the style and color of an element's border.

The `border-style` property specifies what kind of border to display. For example, `inset`: Defines a 3D inset border while `outset` defines a 3D outset border. The effect depends on the `border-color` value

See CSS: border

## 6.

csv.js

```
// See http://en.wikipedia.org/wiki/Comma-separated_values
"use strict"; // Use ECMAScript 5 strict mode in browsers that support it

$(document).ready(function() {
 $("button").click(function() {
 calculate();
 });
});

function calculate() {
 var result;
```

```

var original = document.getElementById("original");
var temp = original.value;
var regexp = /-----/g;
var lines = temp.split(/\n\s*/);
var commonLength = NaN;
var r = [];
// Template using underscore
var row = "<%% _.each(items, function(name) { %>" +
 " <td><%%= name %></td>" +
 " <%% }>); %>";

if (window.localStorage) localStorage.original = temp;

for(var t in lines) {
 var temp = lines[t];
 var m = temp.match(regexp);
 var result = [];
 var error = false;

 if (m) {
 if (commonLength && (commonLength != m.length)) {
 //alert('ERROR! row <'+temp+'> has '+m.length+' items!');
 error = true;
 }
 else {
 commonLength = m.length;
 error = false;
 }
 for(var i in m) {
 var removecomma = m[i].replace(/,\s*$/, '');
 var remove1stquote = removecomma.replace(/^\"s*/, '');
 var removelastquote = remove1stquote.replace(/\"s*$/, '');
 var removeescapedquotes = removelastquote.replace(/\"/, '');
 result.push(removeescapedquotes);
 }
 var tr = error? '<tr class="error">' : '<tr>';
 r.push(tr+_.template(row, {items : result})+"</tr>");
 }
 else {
 alert('ERROR! row '+temp+' does not look as legal CSV');
 error = true;
 }
}
r.unshift('<p>\n<table class="center" id="result">');
r.push('</table>');
//alert(r.join('\n')); // debug
finaltable.innerHTML = r.join('\n');
}

window.onload = function() {
 // If the browser supports localStorage and we have some stored data
 if (window.localStorage && localStorage.original) {
 document.getElementById("original").value = localStorage.original;
 }
}

```

```
}
};
```

## 1. Tutorials:Getting Started with jQuery

### Tareas

- Añada pruebas usando Mocha y Chai

## 1.4. Comentarios y Consejos

### How can I push a local Git branch to a remote with a different name easily?

```
$ git branch -a
* gh-pages
remotes/origin/HEAD -> origin/gh-pages
remotes/origin/gh-pages
```

Of course a solution for this way to work is to rename your master branch:

```
$ git branch -m master gh-pages
[~/Downloads/tmp(gh-pages)]$ git branch
* gh-pages
```

Otherwise, you can do your initial push this way:

```
$ git push -u origin master:gh-pages
```

Option `-u`: for every branch that is up to date or successfully pushed, add **upstream** (tracking) reference, used by argument-less **git-pull**.

- How can I push a local Git branch to a remote with a different name easily?

### favicons y shortcut icons

- A *favicon* (short for *Favorite icon*), also known as a *shortcut* icon, is a file containing one or more small icons, most commonly 16×16 pixels, associated with a particular Web site or Web page.
- A web designer can create such an icon and install it into a Web site (or Web page) by several means, and graphical web browsers will then make use of it.
- Browsers that provide favicon support typically display a page's favicon in the browser's address bar (sometimes in the history as well) and next to the page's name in a list of bookmarks.
- Browsers that support a tabbed document interface typically show a page's favicon next to the page's title on the tab
- Some services in the cloud to generate favicons:
  - Favicon Generator
  - favicon.cc
- En `index.html` poner una línea como una de estas:

```
<link rel="shortcut icon" href="etsiiull.png" type="image/x-icon">
<link rel="shortcut icon" href="logo.png" />
<link href="images/favicon.ico" rel="icon" type="image/x-icon" />
```

## 1.5. Ejercicios

### 1. Paréntesis:

```
> str = "John Smith"
'John Smith'
> newstr = str.replace(re, "$2, $1")
'Smith, John'
```

### 2. El método exec.

If your regular expression uses the `g` flag, you can use the `exec` method multiple times to find successive matches in the same string. When you do so, the search starts at the substring of `str` specified by the regular expression's `lastIndex` property.

```
> re = /d(b+)(d)/ig
/d(b+)(d)/gi
> z = "dBdxdbbdzdbd"
'dBdxdbbdzdbd'
> result = re.exec(z)
['dBd', 'B', 'd', index: 0, input: 'dBdxdbbdzdbd']
> re.lastIndex
3
> result = re.exec(z)
['dbbd', 'bb', 'd', index: 4, input: 'dBdxdbbdzdbd']
> re.lastIndex
8
> result = re.exec(z)
['dbd', 'b', 'd', index: 9, input: 'dBdxdbbdzdbd']
> re.lastIndex
12
> z.length
12
> result = re.exec(z)
null
```

### 3. JavaScript tiene lookahead:

```
> x = "hello"
'hello'
> r = /l(?=o)/
/l(?=o)/
> z = r.exec(x)
['l', index: 3, input: 'hello']
```

### 4. JavaScript no tiene lookbehinds:

```
> x = "hello"
'hello'
> r = /(?!<=l)l/
SyntaxError: Invalid regular expression: /(?!<=l)l/: Invalid group
> .exit
```

```
[~/Dropbox/src/javascript/PLgrado/csv(master)]$ irb
ruby-1.9.2-head :001 > x = "hello"
=> "hello"
ruby-1.9.2-head :002 > r = /(?!<=1)l/
=> ll
ruby-1.9.2-head :008 > x =~ r
=> 3
ruby-1.9.2-head :009 > $&
=> "l"
```

5. El siguiente ejemplo comprueba la validez de números de teléfono:

```
[~/local/src/javascript/PLgrado/regexp]$ pwd -P
/Users/casiano/local/src/javascript/PLgrado/regexp
[~/local/src/javascript/PLgrado/regexp]$ cat phone.html
<!DOCTYPE html>
<html>
 <head>
 <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
 <meta http-equiv="Content-Script-Type" content="text/javascript">
 <script type="text/javascript">
 var re = /\d{3}\)?([-\./])\d{3}\1\d{4}/;
 function testInfo(phoneInput){
 var OK = re.exec(phoneInput.value);
 if (!OK)
 window.alert(RegExp.input + " isn't a phone number with area code!");
 else
 window.alert("Thanks, your phone number is " + OK[0]);
 }
 </script>
 </head>
 <body>
 <p>Enter your phone number (with area code) and then click "Check".

The expected format is like ###-###-####.</p>
 <form action="#">
 <input id="phone"><button onclick="testInfo(document.getElementById('phone'));">Che
 </form>
 </body>
</html>
```

6. ¿Con que cadenas casa la expresión regular /^(11+)\1+\$/?

```
> '1111'.match(/^(11+)\1+$/) # 4 unos
['1111',
 '11',
 index: 0,
 input: '1111']
> '111'.match(/^(11+)\1+$/) # 3 unos
null
> '11111'.match(/^(11+)\1+$/) # 5 unos
null
> '111111'.match(/^(11+)\1+$/) # 6 unos
['111111',
 '111',
 index: 0,
 input: '111111']
```

```

 index: 0,
 input: '111111']
> '11111111'.match(/^(11+)\1+$/) # 8 unos
['11111111',
 '1111',
 index: 0,
 input: '11111111']
> '11111111'.match(/^(11+)\1+$/)
null
>

```

Busque una solución al siguiente ejercicio (véase 'Regex to add space after punctuation sign' en PerlMonks) Se quiere poner un espacio en blanco después de la aparición de cada coma:

```

7. > x = "a,b,c,1,2,d, e,f"
 'a,b,c,1,2,d, e,f'
> x.replace(/,/g, " ")
 'a, b, c, 1, 2, d, e, f'

```

pero se quiere que la sustitución no tenga lugar si la coma esta incrustada entre dos dígitos. Además se pide que si hay ya un espacio después de la coma, no se duplique.

a) La siguiente solución logra el segundo objetivo, pero estropea los números:

```

> x = "a,b,c,1,2,d, e,f"
 'a,b,c,1,2,d, e,f'
> x.replace(/,(\S)/g, " $1")
 'a, b, c, 1, 2, d, e, f'

```

b) Esta otra funciona bien con los números pero no con los espacios ya existentes:

```

> x = "a,b,c,1,2,d, e,f"
 'a,b,c,1,2,d, e,f'
> x.replace(/,(\D)/g, " $1")
 'a, b, c,1,2, d, e, f'

```

c) Explique cuando casa esta expresión regular:

```

> r = /(\d[,.\]\d)|\((?=\S))/g
/(\d[,.\]\d)|\((?=\S))/g

```

Aproveche que el método `replace` puede recibir como segundo argumento una función (vea `replace`):

```

> z = "a,b,1,2,d, 3,4,e"
 'a,b,1,2,d, 3,4,e'
> f = function(match, p1, p2, offset, string) { return (p1 || p2 + " "); }
[Function]
> z.replace(r, f)
 'a, b, 1,2, d, 3,4, e'

```

## 1.6. Práctica: Palabras Repetidas

Se trata de producir una salida en las que las palabras repetidas consecutivas sean reducidas a una sola aparición. Rellena las partes que faltan.

## Donde

```
[~/srcPLgrado/repeatedwords(master)]$ pwd -P
/Users/casiano/local/src/javascript/PLgrado/repeatedwords
[~/srcPLgrado/repeatedwords(master)]$ git remote -v
origin ssh://git@bitbucket.org/casiano/pl-grado-repeated-words.git (fetch)
origin ssh://git@bitbucket.org/casiano/pl-grado-repeated-words.git (push)
```

Véase: <https://bitbucket.org/casiano/pl-grado-repeated-words>

## Ejemplo de ejecución

### Estructura

#### index.html

```
[~/Dropbox/src/javascript/PLgrado/repeatedwords(master)]$ cat index.html
<html>
 <head>
 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
 <title>File Input</title>
 <link href="global.css" rel="stylesheet" type="text/css">

 <script type="text/javascript" src="../../underscore/underscore.js"></script>
 <script type="text/javascript" src="../../jquery/starterkit/jquery.js"></script>
 <script type="text/javascript" src="repeated_words.js"></script>
 </head>
 <body>
 <h1>File Input</h1>
 <input type="file" id="fileinput" />
 <div id="out" class="hidden">
 <table>
 <tr><th>Original</th><th>Transformed</th></tr>
 <tr>
 <td>
 <pre class="input" id="initialinput"></pre>
 </td>
 <td>
 <pre class="output" id="finaloutput"></pre>
 </td>
 </tr>
 </table>
 </div>
 </body>
</html>
```

#### 1. Tag input

#### global.css

Rellena los estilos para `hidden` y `unhidden`:

```
[~/Dropbox/src/javascript/PLgrado/repeatedwords(master)]$ cat global.css
html *
{
```

```

font-size: large;
/* The !important ensures that nothing can override what you've set in this style
(unless it is also important). */
font-family: Arial;
}

.thumb {
 height: 75px;
 border: 1px solid #000;
 margin: 10px 5px 0 0;
}

h1 { text-align: center; font-size: x-large; }
th, td { vertical-align: top; text-align: right; }
/* #finaltable * { color: white; background-color: black; } */

/* #finaltable table { border-collapse: collapse; } */
/* #finaltable table, td { border: 1px solid white; } */
#finaltable:hover td { background-color: blue; }
tr:nth-child(odd) { background-color: #eee; }
tr:nth-child(even) { background-color: #00FF66; }
input { text-align: right; border: none; } /* Align input to the right */
textarea { border: outset; border-color: white; }
table { border: inset; border-color: white; }
.hidden { display: none; }
.unhidden { display: inline-block; }
table.center { margin-left: auto; margin-right: auto; }
#result { border-color: red; }
tr.error { background-color: red; }
pre.output { background-color: white; }
span.repeated { background-color: red }

body
{
 background-color: #b0c4de; /* blue */

```

## 1. CSS display Property

## 2. Diferencias entre "Display" y "Visibility"

### repeated\_words.js

Rellenar las expresiones regulares que faltan:

```

[~/srcPLgrado/repeatedwords(master)]$ cat repeated_words.js
"use strict"; // Use ECMAScript 5 strict mode in browsers that support it

$(document).ready(function() {
 $("#fileinput").change(calculate);
});

function generateOutput(contents) {
 return contents.replace(/_____/g, '_____');
}

```



```

function calculate(evt) {
 var f = evt.target.files[0];
 var contents = '';

 if (f) {
 var r = new FileReader();
 r.onload = function(e) {
 contents = e.target.result;
 var escaped = escapeHtml(contents);
 var outdiv = document.getElementById("out");
 outdiv.className = 'unhidden';
 finaloutput.innerHTML = generateOutput(escaped);
 initialinput.innerHTML = escaped;

 }
 r.readAsText(f);
 } else {
 alert("Failed to load file");
 }
}

var entityMap = {
 "&": "&",
 "<": "<",
 ">": ">",
 "'": '"',
 '"': ''',
 "/": '/',
};

function escapeHtml(string) {
 return String(string).replace(/_____/g, function (s) {
 return _____;
 });
}

```

1. jQuery event.target
2. HTML 5 File API
3. HTML 5 File API: FileReader
4. HTML 5 File API: FileReader
5. element.className
6. HTML Entities
7. Tutorials:Getting Started with jQuery
8. Underscore: template

## Ficheros de Entrada

```

[~/Dropbox/src/javascript/PLgrado/repeatedwords(master)]$ cat input2.txt
habia una vez
vez un viejo viejo

```

```

hidalgo que vivia
vivia
[~/Dropbox/src/javascript/PLgrado/repeatedwords(master)]$ cat input.txt
one one
nothing rep
is two three
three four
[~/Dropbox/src/javascript/PLgrado/repeatedwords(master)]$ cat inputhtml1.txt
habia => una vez
vez & un viejo viejo <puchum>
hidalgo & <pacham> que vivia
vivia </que se yo>

```

## 1.7. Ejercicios

El formato *INI* es un formato estandar para la escritura de ficheros de configuración. Su estructura básica se compone de "secciones" y "propiedades". Véase la entrada de la wikipedia INI.

```

; last modified 1 April 2001 by John Doe
[owner]
name=John Doe
organization=Acme Widgets Inc.

[database]
; use IP address in case network name resolution is not working
server=192.0.2.62
port=143
file = "payroll.dat"

```

1. Escriba un programa javascript que obtenga las cabeceras de sección de un fichero INI
2. Escriba un programa javascript que case con los bloques de un fichero INI (cabecera mas lista de pares `parámetro=valor`)
3. Se quieren obtener todos los pares nombre-valor, usando paréntesis con memoria para capturar cada parte.
4. ¿Que casa con cada paréntesis en esta regexp para los pares nombre-valor?

```

> x = "h = 4"
> r = /([^\s]*)=(\s*)(.*)/
> r.exec(x)
>

```

## 1.8. Ejercicios

1. Escriba una expresión regular que reconozca las cadenas de doble comillas. Debe permitir la presencia de comillas y caracteres escapados.
2. ¿Cual es la salida?

```

> "bb".match(/b|bb/)

> "bb".match(/bb|b/)

```

## 1.9. Práctica: Ficheros INI

### Donde

```
[~/srcPLgrado/ini(develop)]$ pwd -P
/Users/casiano/local/src/javascript/PLgrado/ini
[~/srcPLgrado/ini(develop)]$ git remote -v
origin ssh://git@bitbucket.org/casiano/pl-grado-ini-files.git (fetch)
origin ssh://git@bitbucket.org/casiano/pl-grado-ini-files.git (push)
```

Véase

- Repositorio conteniendo el código (inicial) del analizador de ficheros ini: <https://github.com/crguezl/pl-grado-ini-files>
- Despliegue en GitHub pages: <http://crguezl.github.io/pl-grado-ini-files/>
- Repositorio privado del profesor: <https://bitbucket.org/casiano/pl-grado-ini-files/src>.

### index.html

```
[~/javascript/PLgrado/ini(master)]$ cat index.html
<html>
 <head>
 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
 <title>INI files</title>
 <link href="global.css" rel="stylesheet" type="text/css">
 <!--
 <link rel="shortcut icon" href="logo.png" />
 -->
 <link rel="shortcut icon" href="etsiiull.png" type="image/x-icon">

 <script type="text/javascript" src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/
 <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.10.2/jquery.min.js"></script>

 <script type="text/javascript" src="ini.js"></script>
 </head>
 <body>
 <h1>INI files</h1>
 <input type="file" id="fileinput" />
 <div id="out" class="hidden">
 <table>
 <tr><th>Original</th><th>Tokens</th></tr>
 <tr>
 <td>
 <pre class="input" id="initialinput"></pre>
 </td>
 <td>
 <pre class="output" id="finaloutput"></pre>
 </td>
 </tr>
 </table>
 </div>
 </body>
</html>
```

## Ficheros

Vease

- Reading files in JavaScript using the File APIs by Eric Bidelman.

Source code in files-in-javascript-tut

- W3C File API

- Ejemplo `FileList` en

- github
- en acción en gh-pages.
- Tambien en jsfiddle
- o bien

```
[~/src/javascript/fileapi/html5rocks]$ pwd -P
/Users/casiano/local/src/javascript/fileapi/html5rocks
[~/src/javascript/fileapi/html5rocks(master)]$ ls -l filelist.html
-rw-r--r-- 1 casiano staff 767 15 feb 17:21 filelist.html
```

- The `EventTarget.addEventListener()` method

```
target.addEventListener(type, listener[, useCapture]);
```

registers the specified listener on the `EventTarget` it's called on. The event target may be an `Element` in a document, the `Document` itself, a `Window`, or any other object that supports events (such as `XMLHttpRequest`).

- ```
> date = new Date(Date.UTC(2012, 11, 12, 3, 0, 0));
Wed Dec 12 2012 03:00:00 GMT+0000 (WET)
> date.toLocaleDateString()
"12/12/2012"
```

- `Date.prototype.toLocaleDateString()`

- Ejemplo de Drag and Drop en

- GitHub
- gh-pages
- jsfiddle

o bien en:

```
[~/src/javascript/fileapi/html5rocks]$ pwd -P
/Users/casiano/local/src/javascript/fileapi/html5rocks
[~/src/javascript/fileapi/html5rocks]$ ls -l dragandrop.html
-rw-r--r--  1 casiano  staff  1535 15 feb 18:25 dragandrop.html
```

- `stopPropagation` stops the event from bubbling up the event chain.

Suppose you have a table and within that table you have an anchor tag. Both the table and the anchor tag have code to handle mouse clicks. When the user clicks on the anchor tag, which HTML element should process the event first? Should it be the table then the anchor tag or vice versa?

Formally, the event path is broken into three phases.

- In the *capture phase*, the event starts at the top of the DOM tree, and propagates through to the parent of the target.
- In the *target phase*, the event object arrives at its target. This is generally where you will write your event-handling code.
- In the *bubble phase*, the event will move back up through the tree until it reaches the top. Bubble phase propagation happens in reverse order to the capture phase, with an event starting at the parent of the target and ending up back at the top of the DOM tree.
- jsfiddle

These days, there's a choice to register an event in either the capture phase or the bubble phase. If you register an event in the capture phase, the parent element will process the event before the child element.

- `preventDefault` prevents the default action the browser makes on that event.
- After you've obtained a `File` reference, instantiate a `FileReader` object to read its contents into memory.

```
var reader = new FileReader();
```

to read the file we call one of the `readAs...` For example `readAsDataURL` is used to start reading the contents of the specified `Blob` or `File`:

```
reader.readAsDataURL(f);
```

- Methods to remember:
 - `FileReader.abort()` Aborts the read operation. Upon return, the `readyState` will be `DONE`.
 - `FileReader.readAsArrayBuffer()` Starts reading the contents of the specified `Blob`, once finished, the `result` attribute contains an `ArrayBuffer` representing the file's data.
 - `FileReader.readAsBinaryString()` Starts reading the contents of the specified `Blob`, once finished, the `result` attribute contains the raw binary data from the file as a string.
 - `FileReader.readAsDataURL()` Starts reading the contents of the specified `Blob`.
When the read operation is finished, the `readyState` becomes `DONE`, and the `loadend` is triggered. At that time, the `result` attribute contains a URL representing the file's data as base64 encoded string.
 - `FileReader.readAsText()` Starts reading the contents of the specified `Blob`, once finished, the `result` attribute contains the contents of the file as a text string.

Once one of these read methods is called on your `FileReader` object, the `onloadstart`, `onprogress`, `onload`, `onabort`, `onerror`, and `onloadend` can be used to track its progress.

- When the load finishes, the reader's `onload` event is fired and its `result` attribute can be used to access the file data.

```
reader.onload = function(e) {
    var contents = e.target.result;

    ....
}
```

See

- jsfiddle
- GitHub
- gh-pages
- or

```
[~/src/javascript/fileapi/html5rocks]$ pwd -P
/Users/casiano/local/src/javascript/fileapi/html5rocks
[~/src/javascript/fileapi/html5rocks]$ ls -l readimages.html
-rw-r--r--  1 casiano  staff  1530 15 feb 21:00 readimages.html
```

- base64 testing image jsfiddle
- The `insertBefore()` method inserts a node as a child, right before an existing child, which you specify. See

```
[~/src/javascript/fileapi/html5rocks]$ ls -l readimages.html
-rw-r--r--  1 casiano  staff  1530 15 feb 21:00 readimages.html
```

global.css

```
[~/javascript/PLgrado/ini(master)]$ cat global.css
html *
{
    font-size: large;
    /* The !important ensures that nothing can override what you've set in this style (unless i
    font-family: Arial;
}

.thumb {
    height: 75px;
    border: 1px solid #000;
    margin: 10px 5px 0 0;
}

h1          { text-align: center; font-size: x-large; }
th, td      { vertical-align: top; text-align: left; }
/* #finaltable * { color: white; background-color: black; } */

/* #finaltable table { border-collapse: collapse; } */
/* #finaltable table, td { border: 1px solid white; } */
#finaltable:hover td { background-color: blue; }
tr:nth-child(odd)    { background-color: #eee; }
tr:nth-child(even)   { background-color: #00FF66; }
input               { text-align: right; border: none; } /* Align input to the right */
textarea           { border: outset; border-color: white; }
table              { border: inset; border-color: white; }
.hidden            { display: none; }
.unhidden          { display: block; }
table.center       { margin-left: auto; margin-right: auto; }
#result            { border-color: red; }
tr.error           { background-color: red; }
```

```
pre.output { background-color: white; }
/*
span.repeated { background-color: red }
span.header { background-color: blue }
span.comments { background-color: orange }
span.blanks { background-color: green }
span.nameEqualValue { background-color: cyan }
span.error { background-color: red }
*/
body
{
  background-color:#b0c4de; /* blue */
}
```

Ficheros de Prueba

```
~/Dropbox/src/javascript/PLgrado/ini(master)]$ cat input.ini
```

```
; last modified 1 April 2001 by John Doe
```

```
[owner]
```

```
name=John Doe
```

```
organization=Acme Widgets Inc.
```

```
[database]
```

```
; use IP address in case network name resolution is not working
```

```
server=192.0.2.62
```

```
port=143
```

```
file = "payroll.dat"
```

```
$ cat input2.ini
```

```
[special_fields]
```

```
required = "EmailAddr,FirstName,LastName,Mesg"
```

```
csvfile = "contacts.csv"
```

```
csvcolumns = "EmailAddr,FirstName,LastName,Mesg,Date,Time"
```

```
[email_addresses]
```

```
sales = "jack@yahoo.com,mary@my-sales-force.com,president@my-company.com"
```

```
$ cat inputerror.ini
```

```
[owner]
```

```
name=John Doe
```

```
organization$Acme Widgets Inc.
```

```
[database]
```

```
; use IP address in case network name resolution is not working
```

```
server=192.0.2.62
```

```
port=143
```

```
file = "payroll.dat"
```

ini.js

```
[~/javascript/PLgrado/ini(master)]$ cat ini.js
```

```
"use strict"; // Use ECMAScript 5 strict mode in browsers that support it
```

```
$(document).ready(function() {
```

```

    $("#fileinput").change(calculate);
});

function calculate(evt) {
    var f = evt.target.files[0];

    if (f) {
        var r = new FileReader();
        r.onload = function(e) {
            var contents = e.target.result;

            var tokens = lexer(contents);
            var pretty = tokensToString(tokens);

            out.className = 'unhidden';
            initialinput.innerHTML = contents;
            finaloutput.innerHTML = pretty;
        }
        r.readAsText(f);
    } else {
        alert("Failed to load file");
    }
}

var temp = '<li> <span class = "<%= token.type %>"> <%= match %> </span>\n';

function tokensToString(tokens) {
    var r = '';
    for(var i=0; i < tokens.length; i++) {
        var t = tokens[i]
        var s = JSON.stringify(t, undefined, 2);
        s = _.template(temp, {token: t, match: s});
        r += s;
    }
    return '<ol>\n'+r+'</ol>';
}

function lexer(input) {
    var blanks      = /^\\s+\\/;
    var iniheader   = /^\\[([\\^\\]\\\\r\\\\n)+\\]\\/;
    var comments    = /^\\[;#\\](.*)\\/;
    var nameEqualValue = /^\\([\\^=;\\\\r\\\\n]+\\)=([\\^;\\\\r\\\\n]*)\\/;
    var any         = /^\\.|\\\\n+\\/;

    var out = [];
    var m = null;

    while (input != '') {
        if (m = blanks.exec(input)) {
            input = input.substr(m.index+m[0].length);
            out.push({ type : 'blanks', match: m });
        }
        else if (m = iniheader.exec(input)) {

```



```

    input = input.substr(m.index+m[0].length);
    out.push({ type: 'header', match: m });
  }
  else if (m = comments.exec(input)) {
    input = input.substr(m.index+m[0].length);
    out.push({ type: 'comments', match: m });
  }
  else if (m = nameEqualValue.exec(input)) {
    input = input.substr(m.index+m[0].length);
    out.push({ type: 'nameEqualValue', match: m });
  }
  else if (m = any.exec(input)) {
    out.push({ type: 'error', match: m });
    input = '';
  }
  else {
    alert("Fatal Error!" + substr(input,0,20));
    input = '';
  }
}
return out;
}

```

Véase la sección *JSON.stringify()* 1.3 para saber mas sobre `JSON.stringify`.

Dudas sobre la Sintáxis del Formato INI La sintáxis de INI no está bien definida. Se aceptan decisiones razonables para cada una de las expresiones regulares. Si quiere ver un parser en acción puede instalar la gema `inifile` (Ruby).

Una opción que no hemos contemplado en nuestro código es la posibilidad de hacer que una línea de asignación se expanda en varias líneas. En `inifile` el carácter `\` indica que la línea continúa en la siguiente:

```

[~/javascript/PLgrado/inifile(master)]$ cat test/data/good.ini
[section_one]
one = 1
two = 2

[section_two]
three =      3
multi = multiline \
support

; comments should be ignored
[section three]
four  =4
five=5
six =6

[section_four]
  [section_five]
    seven and eight= 7 & 8

[~/javascript/PLgrado/inifile(master)]$ pry
[2] pry(main)> require 'inifile'
=> true

```

```
[3] pry(main)> p = IniFile.new(:filename => 'test/data/good.ini')
=> #<IniFile:0x007fba2f41a500
  @_line=" seven and eight= 7 & 8",
  @_section={"seven and eight"=>"7 & 8"},
  @comment=";#",
  @content=
    "[section_one]\none = 1\ntwo = 2\n\n[section_two]\nthree =          3\nmulti = multiline \\n
  @default="global",
  @encoding=nil,
  @escape=true,
  @filename="test/data/good.ini",
  @ini=
    {"section_one"=>{"one"=>"1", "two"=>"2"},
     "section_two"=>{"three"=>"3", "multi"=>"multiline support"},
     "section three"=>{"four"=>"4", "five"=>"5", "six"=>"6"},
     "section_four"=>{},
     "section_five"=>{"seven and eight"=>"7 & 8"}},
  @param=""=>

[4] pry(main)> p["section_two"]
=> {"three"=>"3", "multi"=>"multiline support"}
[5] pry(main)> p[:section_two]
```

Tareas

Es conveniente que consiga estos objetivos:

- Pueden comenzar haciendo un fork del repositorio <https://github.com/crguezl/pl-grado-ini-files>.
- La entrada debería poder leerse desde un fichero. Añada drag and drop.
- Use Web Storage igual que en la anterior
- Escriba las pruebas
- Use templates externos `underscore` para estructurar la salida
- Añada soporte para multilíneas en las asignaciones (Véase la sección 1.9)

```
> s = 'a=b\\nc'
'a=b\\nc'
> n2 = /^( [=;#\r\n]+)=((?:[ ^;#\r\n]*\\n)*[ ^;#\r\n]*)/
/^( [=;#\r\n]+)=((?:[ ^;#\r\n]*\\n)*[ ^;#\r\n]*)/
> m = n2.exec(s)
[ 'a=b\\nc', 'a', 'b\\nc',
  index: 0, input: 'a=b\\nc' ]
> d = m[2]
'b\\nc'
> d.replace(/\\n/g, ' ')
'b c'
```

Véase

1. `JSON.stringify`
2. www.json.org
3. `JSON` in JavaScript

4. Underscore: template
5. Stackoverflow::how to use Underscore template

1.10. Práctica: Analizador Léxico para Un Subconjunto de JavaScript

TDOP, Top Down Operator Precedence Vamos a trabajar a partir de este repo de Douglas Crockford:

- <https://github.com/douglascrockford/TDOP>
- Autor: Douglas Crockford, douglas@crockford.com
- Fecha que figura en el repo: 2010-11-12
- Descripción:
 - `tdop.html` contains a description of Vaughn Pratt's Top Down Operator Precedence, and describes the parser whose lexer we are going to write in this lab. Is a simplified version of JavaScript.
 - The file `index.html` parses `parse.js` and displays its AST.
 - The page depends on `parse.js` and `tokens.js`.
 - The file `tdop.js` contains the Simplified JavaScript parser.
 - `tokens.js`. produces an array of token objects from a string. This is the file we are going to work in this lab.

Objetivos de la Práctica

Douglas Crockford escribió su analizador léxico `tokens.js` sin usar expresiones regulares. Eso hace que sea extenso (268 líneas). Su analizador es un subconjunto de JS que no tiene - entre otras cosas - expresiones regulares y uno de sus objetivos era que el analizador se analizara a si mismo.

Reescriba el analizador léxico en `tokens.js`. usando expresiones regulares.

1. Evite que se hagan copias de la cadena siendo procesada. Muévase dentro de la misma cadena usando `lastIndex`
2. Añada botones/enlaces/menu de selección que permitan cargar un fichero específico de una lista de ficheros en la `texarea` de entrada.

Vea el ejemplo en <https://github.com/crguezl/loadfileontotexarea>.

En este caso en vez de un fichero `index.html` arrancamos desde un programa Ruby `app.rb`. Para verlo en ejecución instale primero las dependencias:

```
[~/javascript/jquery/loadfileontotexarea(master)]$ bundle install
Using daemons (1.1.9)
Using eventmachine (1.0.3)
Using rack (1.5.2)
Using rack-protection (1.5.2)
Using tilt (1.4.1)
Using sinatra (1.4.4)
Using thin (1.6.1)
Using bundler (1.3.5)
Your bundle is complete!
Use 'bundle show [gemname]' to see where a bundled gem is installed.
```

Para ejecutar puede llamar a la aplicación así:

```
[~/javascript/jquery/loadfileontotexarea(master)]$ bundle exec rackup
Thin web server (v1.6.1 codename Death Proof)
Maximum connections set to 1024
Listening on 0.0.0.0:9292, CTRL+C to stop
```

Ahora visite en su navegador la URL `http://localhost:9292`.

Puede ver también la aplicación corriendo en los servidores de Heroku en `http://pllexer.herokuapp.com/`.
Visite los enlaces `withajax.html` y `withget.html`.

3. Añada pruebas
4. Haga el despliegue de su aplicación en Heroku. Para ver como hacerlo siga las indicaciones en la sección *Heroku* 12 en estos apuntes
5. Una primera solución de la que puede partir se encuentra en: <https://github.com/crguezl/ull-etsii-grado-pl-mini-javascript/tree/gh-pages> en github. Veala en funcionamiento en GitHub Pages
6. El método `tokens` retorna el array de tokens. Puede encontrarlo en `tokens.js`.
7. Mejore la solución en <https://github.com/crguezl/ull-etsii-grado-pl-mini-javascript/tree/gh-pages>
8. Para esta práctica es necesario familiarizarse con la forma en que funciona la OOP en JS. Ve a este jsfiddle

Capítulo 2

Análisis Sintáctico Mediante Precedencia de Operadores en JavaScript

2.1. Ejemplo Simple de Intérprete: Una Calculadora

1. How to write a simple interpreter in JavaScript

2.2. Análisis Top Down Usando Precedencia de Operadores

1. Véase el libro [2] Beautiful Code: Leading Programmers Explain How They Think, Capítulo 9.
2. Top Down Operator Precedence por Douglas Crockford
3. Top Down Operator Precedence demo por Douglas Crockford
4. jslint
5. David Majda - Easy parsing with PEG.js

2.2.1. Gramática de JavaScript

1. Especificación de JavaScript 1997
2. NQLL(1) grammar (Not Quite LL(1)) for JavaScript 1997
3. Postscript con la especificación de JavaScript 1997
4. Mozilla JavaScript Language Resources
5. JavaScript 1.4 LR(1) Grammar 1999.
6. Apple JavaScript Core Specifications
7. Creating a JavaScript Parser Una implementación de ECMAScript 5.1 usando Jison disponible en GitHub en <https://github.com/cjihrig/jsparser>.

Capítulo 3

Análisis Descendente mediante Parsing Expresion Grammars en JavaScript

3.1. Conceptos Básicos para el Análisis Sintáctico

Suponemos que el lector de esta sección ha realizado con éxito un curso en teoría de autómatas y lenguajes formales. Las siguientes definiciones repasan los conceptos mas importantes.

Definición 3.1.1. Dado un conjunto A , se define A^* el cierre de Kleene de A como: $A^* = \bigcup_{n=0}^{\infty} A^n$. Se admite que $A^0 = \{\epsilon\}$, donde ϵ denota la palabra vacía, esto es la palabra que tiene longitud cero, formada por cero símbolos del conjunto base A .

Definición 3.1.2. Una gramática G es una cuaterna $G = (\Sigma, V, P, S)$. Σ es el conjunto de terminales. V es un conjunto (disjunto de Σ) que se denomina conjunto de variables sintácticas o categorías gramaticales, P es un conjunto de pares de $V \times (V \cup \Sigma)^*$. En vez de escribir un par usando la notación $(A, \alpha) \in P$ se escribe $A \rightarrow \alpha$. Un elemento de P se denomina producción. Por último, S es un símbolo del conjunto V que se denomina símbolo de arranque.

Definición 3.1.3. Dada una gramática $G = (\Sigma, V, P, S)$ y $\mu = \alpha A \beta \in (V \cup \Sigma)^*$ una frase formada por variables y terminales y $A \rightarrow \gamma$ una producción de P , decimos que μ deriva en un paso en $\alpha \gamma \beta$. Esto es, derivar una cadena $\alpha A \beta$ es sustituir una variable sintáctica A de V por la parte derecha γ de una de sus reglas de producción. Se dice que μ deriva en n pasos en δ si deriva en $n - 1$ pasos en una cadena $\alpha A \beta$ la cual deriva en un paso en δ . Se escribe entonces que $\mu \xRightarrow{*} \delta$. Una cadena deriva en 0 pasos en si misma.

Definición 3.1.4. Dada una gramática $G = (\Sigma, V, P, S)$ se denota por $L(G)$ o lenguaje generado por G al lenguaje:

$$L(G) = \{x \in \Sigma^* : S \xRightarrow{*} x\}$$

Esto es, el lenguaje generado por la gramática G esta formado por las cadenas de terminales que pueden ser derivados desde el símbolo de arranque.

Definición 3.1.5. Una derivación que comienza en el símbolo de arranque y termina en una secuencia formada por sólo terminales de Σ se dice completa.

Una derivación $\mu \xRightarrow{*} \delta$ en la cual en cada paso $\alpha A x$ la regla de producción aplicada $A \rightarrow \gamma$ se aplica en la variable sintáctica mas a la derecha se dice una derivación a derechas

Una derivación $\mu \xRightarrow{*} \delta$ en la cual en cada paso $x A \alpha$ la regla de producción aplicada $A \rightarrow \gamma$ se aplica en la variable sintáctica mas a la izquierda se dice una derivación a izquierdas

Definición 3.1.6. Observe que una derivación puede ser representada como un árbol cuyos nodos están etiquetados en $V \cup \Sigma$. La aplicación de la regla de producción $A \rightarrow \gamma$ se traduce en asignar como hijos del nodo etiquetado con A a los nodos etiquetados con los símbolos $X_1 \dots X_n$ que constituyen la frase $\gamma = X_1 \dots X_n$. Este árbol se llama árbol sintáctico concreto asociado con la derivación.

Definición 3.1.7. Observe que, dada una frase $x \in L(G)$ una derivación desde el símbolo de arranque da lugar a un árbol. Ese árbol tiene como raíz el símbolo de arranque y como hojas los terminales $x_1 \dots x_n$ que forman x . Dicho árbol se denomina árbol de análisis sintáctico concreto de x . Una derivación determina una forma de recorrido del árbol de análisis sintáctico concreto.

Definición 3.1.8. Una gramática G se dice ambigua si existe alguna frase $x \in L(G)$ con al menos dos árboles sintácticos. Es claro que esta definición es equivalente a afirmar que existe alguna frase $x \in L(G)$ para la cual existen dos derivaciones a izquierda (derecha) distintas.

3.1.1. Ejercicio

Dada la gramática con producciones:

```

program → declarations statements | statements
declarations → declaration ';' declarations | declaration ';'
declaration → INT idlist | STRING idlist
statements → statement ';' statements | statement
statement → ID '=' expression | P expression
expression → term '+' expression | term
term → factor '*' term | factor
factor → '(' expression ')' | ID | NUM | STR
idlist → ID ',' idlist | ID

```

En esta gramática, Σ esta formado por los caracteres entre comillas simples y los símbolos cuyos identificadores están en mayúsculas. Los restantes identificadores corresponden a elementos de V . El símbolo de arranque es $S = \text{program}$.

Conteste a las siguientes cuestiones:

1. Describa con palabras el lenguaje generado.
2. Construya el árbol de análisis sintáctico concreto para cuatro frases del lenguaje.
3. Señale a que recorridos del árbol corresponden las respectivas derivaciones a izquierda y a derecha en el apartado 2.
4. ¿Es ambigua esta gramática?. Justifique su respuesta.

3.2. Análisis Sintáctico Predictivo Recursivo

La siguiente fase en la construcción del analizador es la fase de análisis sintáctico. Esta toma como entrada el flujo de terminales y construye como salida el árbol de análisis sintáctico abstracto.

El árbol de análisis sintáctico abstracto es una representación compactada del árbol de análisis sintáctico concreto que contiene la misma información que éste.

Existen diferentes métodos de análisis sintáctico. La mayoría caen en una de dos categorías: ascendentes y descendentes. Los ascendentes construyen el árbol desde las hojas hacia la raíz. Los descendentes lo hacen en modo inverso. El que describiremos aquí es uno de los mas sencillos: se denomina método de análisis predictivo descendente recursivo.

3.2.1. Introducción

En este método se asocia una subrutina con cada variable sintáctica $A \in V$. Dicha subrutina (que llamaremos A) reconocerá el lenguaje generado desde la variable A :

$$L_A(G) = \{x \in \Sigma^* : A \xRightarrow{*} x\}$$

En este método se escribe una rutina **A** por variable sintáctica $A \in V$. Se le da a la rutina asociada el mismo nombre que a la variable sintáctica asociada. La función de la rutina **A** asociada con la variable $A \in V$ es reconocer el lenguaje $L(A)$ generado por A . La estrategia general que sigue la rutina **A** para reconocer $L(A)$ es decidir en términos del terminal a en la entrada que regla de producción concreta $A \rightarrow \alpha$ se aplica para a continuación comprobar que la entrada que sigue pertenece al lenguaje generado por α . En un analizador predictivo descendente recursivo (APDR) se asume que el símbolo que actualmente esta siendo observado (denotado **lookahead**) permite determinar unívocamente que producción de A hay que aplicar. Una vez que se ha determinado que la regla por la que continuar la derivación es $A \rightarrow \alpha$ se procede a reconocer $L_\alpha(G)$, el lenguaje generado por α . Si $\alpha = X_1 \dots X_n$, las apariciones de terminales X_i en α son emparejadas con los terminales en la entrada mientras que las apariciones de variables $X_i = B$ en α se traducen en llamadas a la correspondiente subrutina asociada con **B**.

Para ilustrar el método, simplificaremos la gramática presentada en el ejercicio 4.1.1 eliminando las declaraciones:

```
statements → statement ';' statements | statement
statement → ID '=' expression | P expression
expression → term '+' expression | term
term → factor '*' term | factor
factor → '(' expression ')' | ID | NUM
```

La secuencia de llamadas cuando se procesa la entrada mediante el siguiente programa construye “implícitamente” el árbol de análisis sintáctico concreto.

Dado que estamos usando **strict** se requiere prototipar las funciones al comienzo del fichero:

```
sub parse();
sub statements();
sub statement();
sub expression();
sub term();
sub factor();
sub idlist();
sub declaration();
sub declarations();
```

Para saber mas sobre prototipos consulte [?].

```
Programa 3.2.1. 1 sub match {
2   my $t = shift;
3
4   if ($lookahead eq $t) {
5     ($lookahead, $value) = splice @tokens,0,2;
6     if (defined($lookahead)) {
7       $lookahead = $value if ($lookahead eq 'PUN');
8     } else { $lookahead = 'EOI'; }
9   }
10  else { error("Se esperaba $t y se encontro $lookahead\n"); }
11 }
12
13 sub statement {
14   if ($lookahead eq 'ID') { match('ID'); match('='); expression; }
15   elsif ($lookahead eq 'P') { match('P'); expression; }
16   else { error('Se esperaba un identificador'); }
17 }
18
19 sub term() {
```



```

20  factor;
21  if ($lookahead eq '*') { match('*'); term; }
22 }
23
24 sub expression() {
25  term;
26  if ($lookahead eq '+') { match('+'); expression; }
27 }
28
29 sub factor() {
30  if ($lookahead eq 'NUM') { match('NUM'); }
31  elsif ($lookahead eq 'ID') { match('ID'); }
32  elsif ($lookahead eq '(') { match('('); expression; match(')'); }
33  else { error("Se esperaba (, NUM o ID"); }
34 }
35
36 sub statements {
37  statement;
38  if ($lookahead eq ';') { match(';'); statements; }
39 }
40
41 sub parser {
42  ($lookahead, $value) = splice @tokens,0,2;
43  statements; match('EOI');
44 }

```

Como vemos en el ejemplo, el análisis predictivo confía en que, si estamos ejecutando la entrada del procedimiento A, el cuál está asociado con la variable $A \in V$, el símbolo terminal que esta en la entrada a determine de manera unívoca la regla de producción $A \rightarrow a\alpha$ que debe ser procesada.

Si se piensa, esta condición requiere que todas las partes derechas α de las reglas $A \rightarrow \alpha$ de A “comiencen” por diferentes símbolos. Para formalizar esta idea, introduciremos el concepto de conjunto $FIRST(\alpha)$:

Definición 3.2.1. Dada una gramática $G = (\Sigma, V, P, S)$ y un símbolo $\alpha \in (V \cup \Sigma)^*$ se define el conjunto $FIRST(\alpha)$ como:

$$FIRST(\alpha) = \left\{ b \in \Sigma : \alpha \xRightarrow{*} b\beta \right\} \cup N(\alpha)$$

donde:

$$N(\alpha) = \begin{cases} \{\epsilon\} & \text{si } \alpha \xRightarrow{*} \epsilon \\ \emptyset & \text{en otro caso} \end{cases}$$

Podemos reformular ahora nuestra afirmación anterior en estos términos: Si $A \rightarrow \gamma_1 \mid \dots \mid \gamma_n$ y los conjuntos $FIRST(\gamma_i)$ son disjuntos podemos construir el procedimiento para la variable A siguiendo este pseudocódigo:

```

sub A {
  if ($lookahead in FIRST(gamma_1)) { imitar gamma_1 }
  elsif ($lookahead in FIRST(gamma_2)) { imitar gamma_2 }
  ...
  else ($lookahead in FIRST(gamma_n)) { imitar gamma_n }
}

```

Donde si γ_j es $X_1 \dots X_k$ el código `gamma_j` consiste en una secuencia $i = 1 \dots k$ de llamadas de uno de estos dos tipos:

- Llamar a la subrutina `X_i` si X_i es una variable sintáctica
- Hacer una llamada a `match(X_i)` si X_i es un terminal

3.2.2. Ejercicio: Recorrido del árbol en un ADPR

¿En que forma es recorrido el árbol de análisis sintáctico concreto en un analizador descendente predictivo recursivo? ¿En que orden son visitados los nodos?

3.2.3. Ejercicio: Factores Comunes

En el programa 3.2.1 el reconocimiento de las categorías gramaticales *statements*, *expression* y *term* (líneas 19-27) difiere del resto. Observe las reglas:

statements \rightarrow *statement* ';' *statements* | *statement*
expression \rightarrow *term* '+' *expression* | *term*
term \rightarrow *factor* '*' *term* | *factor*

¿Son disjuntos los conjuntos $FIRST(\gamma_i)$ para las partes derechas de las reglas de *statements*? ¿Son disjuntos los conjuntos $FIRST(\gamma_i)$ para las partes derechas de las reglas de *expression*? ¿Son disjuntos los conjuntos $FIRST(\gamma_i)$ para las partes derechas de las reglas de *term*?

Si se tiene una variable con producciones:

$A \rightarrow \alpha\beta \mid \alpha\gamma$

Las dos producciones tienen un *máximo factor común* en la izquierda de su parte derecha α . Asumimos que $FIRST(\beta) \cap FIRST(\gamma) = \emptyset$.

1. ¿Cómo puede modificarse la gramática para obtener una nueva gramática que cumpla la condición de que las partes derechas tienen conjuntos $FIRST(\gamma_i)$ disjuntos?
2. ¿Puede modificarse la técnica APDR para que funcione sobre gramáticas con este tipo de producciones?. Observe el código asociado con *statements*, *expression* y *term*. ¿Cómo sería el esquema general?

3.2.4. Derivaciones a vacío

Surge un problema cuando $A \rightarrow \gamma_1 \mid \dots \mid \gamma_n$ y la palabra vacía está en alguno de los conjuntos $FIRST(\gamma_i)$. ¿Que hacer entonces?

Nótese que si $A \rightarrow \gamma$ y $\epsilon \in FIRST(\gamma)$ es porque existe una derivación $\gamma \xRightarrow{*} \epsilon$. ¿Que terminales podemos legalmente encontrarnos cuando estamos en la subrutina **A**? Consideremos una derivación desde el símbolo de arranque en la que se use la producción $A \rightarrow \gamma$. Dicha derivación forzosamente tendrá la forma:

$$S \xRightarrow{*} \beta A \alpha\mu \Rightarrow \beta\gamma \alpha\mu \xRightarrow{*} \beta \alpha\mu.$$

Cualquier terminal $a \in \Sigma$ que pueda aparecer en una derivación desde el símbolo de arranque inmediatamente a continuación de la variable A es susceptible de ser visto cuando se esta analizando A y se aplicó $A \rightarrow \gamma$ con $\gamma \xRightarrow{*} \epsilon$. Esto nos lleva a la definición del conjunto $FOLLOW(A)$ como conjunto de terminales que pueden aparecer a continuación de A en una derivación desde el símbolo de arranque:

Definición 3.2.2. Dada una gramática $G = (\Sigma, V, P, S)$ y una variable $A \in V$ se define el conjunto $FOLLOW(A)$ como:

$$FOLLOW(A) = \left\{ b \in \Sigma : \exists S \xRightarrow{*} \alpha A b \beta \right\} \cup E(A)$$

donde

$$E(A) = \begin{cases} \{\$ \} & \text{si } S \xRightarrow{*} \alpha A \\ \emptyset & \text{en otro caso} \end{cases}$$

Aquí $\$$ denota el final de la entrada (que se corresponde en el código Perl anterior con el terminal EOI).

Si $A \rightarrow \gamma_1 \mid \dots \mid \gamma_n$ dado que los conjuntos $FIRST(\gamma_i)$ han de ser disjuntos para que un analizador predictivo APDR funcione, sólo una parte derecha puede contener la palabra vacía en su $FIRST$.

Supongamos que es γ_n . Podemos reformular la construcción del procedimiento para la variable A siguiendo este pseudocódigo:

```

sub A {
  if ($lookahead in FIRST(gamma_1)) { imitar gamma_1 }
  elsif ($lookahead in FIRST(gamma_2)) { imitar gamma_2 }
  ...
  else ($lookahead in FIRST(gamma_n) or $lookahead in FOLLOW(A)) { imitar gamma_n }
}

```

Un caso particular de $\gamma_n \xrightarrow{*} \epsilon$ es que $\gamma_n = \epsilon$. En tal caso, y como es obvio, el significado de `imitar gamma_n` es equivalente a ejecutar una sentencia vacía.

3.2.5. Construcción de los conjuntos de Primeros y Siguietes

Algoritmo 3.2.1. *Construcción de los conjuntos $FIRST(X)$*

Repita el siguiente conjunto de reglas hasta que no se puedan añadir mas símbolos terminales o a ningún conjunto $FIRST(X)$:

1. Si $X \in \Sigma$ entonces $FIRST(X) = X$
2. Si $X \rightarrow \epsilon$ entonces $FIRST(X) = FIRST(X) \cup \{\epsilon\}$
3. Si $X \in V$ y $X \rightarrow Y_1 Y_2 \cdots Y_k \in P$ entonces

$i = 1;$
hacer
 $FIRST(X) = FIRST(X) \cup FIRST^*(Y_i);$
 $i ++;$
mientras ($i \leq k$ y $\epsilon \in FIRST(Y_i)$)

4. Añadir ϵ a $FIRST(X)$ si $i \geq k$ y $\epsilon \in FIRST(Y_k)$

Aquí $FIRST^*(Y)$ denota al conjunto $FIRST(Y) - \{\epsilon\}$.

Este algoritmo puede ser extendido para calcular $FIRST(\alpha)$ para $\alpha = X_1 X_2 \cdots X_n \in (V \cup \Sigma)^*$. El esquema es análogo al de un símbolo individual.

Algoritmo 3.2.2. *Construcción del conjunto $FIRST(\alpha)$*

Repita siguiente conjunto de reglas hasta que no se puedan añadir mas símbolos terminales o a ningún conjunto $FIRST(\alpha)$:

$i = 1;$
 $FIRST(\alpha) = \emptyset;$
hacer
 $FIRST(\alpha) = FIRST(\alpha) \cup FIRST^*(X_i);$
 $i ++;$
mientras ($i \leq n$ y $\epsilon \in FIRST(X_i)$)

Algoritmo 3.2.3. *Construcción de los conjuntos $FOLLOW(A) \forall A \in V$:*

Repetir los siguientes pasos hasta que ninguno de los conjuntos $FOLLOW$ cambie:

1. $FOLLOW(S) = \{\$ \}$ ($\$$ representa el final de la entrada)
2. Si $A \rightarrow \alpha B \beta$ entonces

$$FOLLOW(B) = FOLLOW(B) \cup (FIRST(\beta) - \{\epsilon\})$$

3. Si $A \rightarrow \alpha B$ o $A \rightarrow \alpha B \beta$ y $\epsilon \in FIRST(\beta)$ entonces

$$FOLLOW(B) = FOLLOW(B) \cup FOLLOW(A)$$

3.2.6. Ejercicio: Construir los *FIRST*

Construya los conjuntos *FIRST* de las partes derechas de las reglas de producción de la gramática presentada en el ejercicio 4.1.1.

3.2.7. Ejercicio: Calcular los *FOLLOW*

Modificamos la gramática de la sección 3.2.1 para que admita la sentencia vacía:

```
statements → statement ';' statements | statement
statement → ID '=' expression | P expression | ε
expression → term '+' expression | term
term → factor '*' term | factor
factor → '(' expression ')' | ID | NUM
```

Calcule los conjuntos *FOLLOW*. ¿Es la nueva gramática susceptible de ser analizada por un analizador predictivo descendente recursivo? ¿Cómo sería el código para la subrutina `statements`?. Escríbalo.

3.2.8. Práctica: Construcción de los *FIRST* y los *FOLLOW*

He escrito un módulo llamado Grammar que provee la función `Grammar::Parse` la cual recibe una cadena conteniendo la gramática en formato `yacc` o `eyapp` y devuelve una referencia a un hash conteniendo la información pertinente para el tratamiento de la gramática. Para instalar el módulo tenga en cuenta que depende del módulo `Parse::Yapp`.

Para ilustrar el uso vea los ejemplos en el directorio `scripts`. En concreto veamos el programa `grammar.pl`.

```
Grammar/scripts$ cat -n grammar.pl
1  #!/usr/bin/perl -w -I../lib
2  use strict;
3  use Grammar;
4  use Data::Dumper;
5
6  sub usage {
7      print <<"EOI";
8  usage:
9  $0 input_grammar
10 EOI
11     die "\n";
12 }
13
14 usage() unless @ARGV;
15 my $filename = shift;
16
17 local $/ = undef;
18 open my $FILE, "$filename";
19 my $grammar = <$FILE>;
20 my $x = Grammar::Parse($grammar);
21
22 print Dumper($x);
```

Vamos a darle como entrada la gramática en el fichero `aSb.yp` conteniendo una gramática:

```
Grammar/scripts$ cat -n aSb.yp
1  %%
2  S:
```

```

3      |  'a' S 'b'
4  ;
5  %%

```

Las gramáticas aceptadas por `Grammar::Parse` se adaptan a la sintaxis de las gramáticas reconocidas por `Parse::Yapp`. Una gramática (normalmente con tipo `.yp`) consta de tres partes: la cabeza, el cuerpo y la cola. Cada una de las partes va separada de las otras por el símbolo `%%` en una línea aparte. Así, el `%%` de la línea 1 separa la cabeza del cuerpo. En la cabecera se colocan las declaraciones de terminales (directiva `%token`), cual es el símbolo de arranque (directiva `%start`), etc. El cuerpo contiene las reglas de la gramática y las acciones asociadas. Por último, la cola en nuestro caso no es usada y es vacía. En general, la cola contiene las rutinas de soporte al código que aparece en las acciones así como, posiblemente, rutinas para el análisis léxico y el tratamiento de errores.

La salida de `Grammar::Parse` es una referencia a un hash cuyas entradas vienen explicadas por los comentarios.

```

Grammar/scripts$ grammar.pl aSb.yp
$VAR1 = {
  'SYMS' => { 'S' => 2, '"b"' => 3, '"a"' => 3 }, # Símbolo => línea
  'NULL' => { 'S' => 1 }, # símbolos que se anulan
  'RULES' => [
    [ 'S', [] ], # S produce vacío
    [ 'S', [ '"a"', 'S', '"b"' ] ] # S -> aSb
  ],
  'START' => 'S', # Símbolo de arranque
  'TERM' => [ '"b"', '"a"' ], # terminales /tokens
  'NTERM' => { 'S' => [ 0, 1 ] } # índices de las reglas de las variables sintácticas
};

```

Usando la estructura devuelta por la función `Grammar::Parse` escriba un módulo que provea funciones para computar los `FIRST` y los `FOLLOW` de las variables sintácticas de la gramática. No olvide escribir la documentación. Incluya una prueba por cada una de las gramáticas que figuran en el directorio `scripts` del módulo `Grammar`.

Puede encontrar la práctica *casi hecha* en `PL::FirstFollow`. Asegúrese de entender el algoritmo usado. Aumente el número de pruebas y haga un análisis de cubrimiento.

3.2.9. Gramáticas LL(1)

Una gramática $G = (\Sigma, V, P, S)$ cuyo lenguaje generado $L(G)$ puede ser analizado por un analizador sintáctico descendente recursivo predictivo se denomina *LL(1)*. Una gramática es LL(1) si y sólo si para cualesquiera dos producciones $A \rightarrow \alpha$ y $A \rightarrow \beta$ de G se cumple:

1. $FIRST(\alpha) \cap FIRST(\beta) = \emptyset$
2. Si $\epsilon \in FIRST(\alpha)$, entonces $FIRST(\alpha) \cap FOLLOW(A) = \emptyset$

¿De donde viene el nombre LL(1)? La primera L hace alusión al hecho de que el flujo de terminales se lee de izquierda a derecha, accediendo a la entrada por su izquierda (*Left*). La segunda L se refiere a que el método de análisis predictivo construye una derivación a izquierdas. El número entre paréntesis indica el número de terminales que debemos consultar para decidir que regla de producción se aplica. Así, en una gramática LL(2) la decisión final de que producción elegir se hace consultando los dos terminales a la entrada.

3.2.10. Ejercicio: Caracterización de una gramática LL(1)

Cuando se dice que una gramática es LL(1) si, y sólo si:

1. $FIRST(\alpha) \cap FIRST(\beta) = \emptyset$

2. Si $\epsilon \in FIRST(\alpha)$, entonces $FIRST(\alpha) \cap FOLLOW(A) = \emptyset$

se asume que los conjuntos $FIRST(\alpha)$ no son vacíos.

- ¿Que se puede decir de la regla $A \rightarrow \alpha$ si $FIRST(\alpha) = \emptyset$?
- ¿Que se puede decir de la variable A si $FOLLOW(A) = \emptyset$?

3.2.11. Ejercicio: Ambigüedad y LL(1)

¿Puede una gramática LL(1) ser ambigua?. Razone su respuesta.

3.2.12. Práctica: Un analizador APDR

Siguiendo con la construcción del compilador para el lenguaje Tutu, escriba un analizador APDR para la siguiente gramática. Reutilice el código de las prácticas de las secciones anteriores (?? y ??).

```

program → declarations statements | statements
declarations → declaration ';' declarations | declaration ';'
declaration → INT idlist | STRING idlist
statements → statement ';' statements | statement
statement → ID '=' expression | P expression | ε
expression → term '+' expression | term
term → factor '*' term | factor
factor → '(' expression ')' | ID | NUM | STR
idlist → ID ',' idlist | ID

```

3.2.13. Práctica: Generación Automática de Analizadores Predictivos

Objetivo Escriba un módulo `GAP.pm` que provea una subrutina `gap` para la generación automática de un APDR supuesto que la gramática de entrada es LL(1).

La subrutina `gap` recibe como entrada la gramática según la estructura de datos generada por la función `Grammar::Parse` de la versión 0.3 del módulo `Grammar`.

El Módulo Grammar La estructura de datos generada por la función `Grammar::Parse` se explicó en la práctica 3.2.8. La estructura ha sido extendida en esta versión para incluir el código que se sitúe en la zona de cola. Por ejemplo, dada la gramática de entrada:

```

Grammar/03/scripts$ cat -n aSb.yp
 1  %%
 2  S:
 3      |   'a' S 'b'
 4  ;
 5  %%
 6
 7  sub Lex {
 8      local $_ = shift; # input
 9      my @tokens;
10
11
12      while ($_) {
13          s/^\s*//; # fuera blancos
14          push @tokens, $1, $1 if s/^(.)/s
15      }
16      @tokens;

```

```

17 }
18
19 sub main {
20     my $filename = shift;
21     my $input;
22
23     if (defined($filename)) {
24         local $/ = undef;
25         open my $FILE, $filename or die "No se pudo abrir $filename\n";
26         $input = <$FILE>;
27         close($FILE);
28     }
29     else { $input = <STDIN> }
30
31     my @tokens = Lex($input);
32     Parse(@tokens); # Llamada al analizador generado
33     print "Sintácticamente correcto\n";
34 }

```

se genera la siguiente estructura de datos:

```

{
  'SYMS' => { 'S' => 2, 'b' => 3, 'a' => 3 }, # Símbolo => línea de aparición
  'NULL' => { 'S' => 1 }, # Símbolos que se anulan
  'RULES' => [ # Reglas
    [ 'S', [] ], # S produce vacío
    [ 'S', [ 'a', 'S', 'b' ] ] # S-> a S b
  ],
  'START' => 'S', # Símbolo de arranque
  'TERM' => [ 'b', 'a' ], # Terminales
  'NTERM' => { 'S' => [ 0, 1 ] } # Variables sintácticas e índices de las reglas de esa variab
  'TAIL' => [ # [ 'Código de cola', línea en la que está el segundo %% ]
,

```

```

sub Lex {
  local $_ = shift; # input
  my @tokens;

  while ($_) {
    s/^\s*//; # fuera blancos
    push @tokens, $1, $1 if s/^(.)/s
  }
  @tokens;
}

```

```

sub main {
  my $filename = shift;
  my $input;

  if (defined($filename)) {
    local $/ = undef;
    open my $FILE, $filename or die "No se pudo abrir $filename\n";
    $input = <$FILE>;

```

```

        close($FILE);
    }
    else { $input = <STDIN> }

    my @tokens = Lex($input);
    my $ok = Parse(@tokens); # Llamada al analizador generado
    print "Sintácticamente correcto\\n" if $ok;
}

', 5 ], # línea en la que está el segundo %%
};

```

Así pues la entrada con clave `TAIL` contiene el código auxiliar de cola. Este código debe ser incluido por su programa dentro del texto del paquete generado por `gap`.

Descripción del objetivo: La función `gap` La función `gap` también recibe como entrada el nombre del package:

```
$package_text = &gap($grammar, 'Package_name');
```

La función `gap` retorna una cadena conteniendo el `package` en el que están las subrutinas del analizador sintáctico.

La idea es que dicha cadena se salvará en un fichero con nombre `Package_name.pm` que podrá posteriormente ser usado (`use Package_name`) por un programa que necesite analizar entradas que se conforman de acuerdo a la especificación de la gramática.

Descripción del objetivo: La función `parser` La rutina principal del paquete generado se ha de llamar `parser` (esto es, su nombre completo es: `Package_name::parser`. Evidentemente `Package_name` debe ser un nombre Perl válido). Ninguna subrutina deberá ser exportada sino que deberán ser llamadas por su nombre completo.

La subrutina `parser` recibe como argumento el array de terminales, obtiene el primer terminal y llama a la subrutina asociada con el símbolo de arranque. Los terminales están representados como parejas (*terminal, atributo*).

Observe que, una vez que la cadena `$package_text` conteniendo el paquete ha sido generada y salvada en un fichero con nombre `Package_name.pm`, podemos escribir un programa cliente:

```

use strict;
use Package_name;

&Package_name::main;

```

Este programa espera una entrada desde fichero o `STDIN` e informa si dicha entrada es sintácticamente correcta o no para la gramática en cuestión.

Cálculo de los `First` y los `Follow` con `PL::FirstFollow` Para facilitar la escritura de `GAP.pm` pueden hacer uso del módulo `PL::FirstFollow` el cual calcula los *FIRST* y los *FOLLOW*. El módulo `PL::FirstFollow` depende de `Set::Scalar` escrito por Jarkko Hietaniemi: instálelo primero.

Deberá familiarizarse con `PL::FirstFollow`, rellenar la documentación de todas las subrutinas (apariciones de `????` en el texto) y escribir la documentación siguiendo el template que se provee. *Rellene los fragmentos de código que se han sustituido por signos de interrogación.* Haga un estudio de cubrimiento y añada pruebas para mejorar el actual. El actual cubrimiento es:

| | | | | | | | |
|-------|-------|-------|-------|-------|-------|-------|-------|
| ----- | ----- | ----- | ----- | ----- | ----- | ----- | ----- |
| File | stmt | bran | cond | sub | pod | time | total |
| ----- | ----- | ----- | ----- | ----- | ----- | ----- | ----- |

| | | | | | | | |
|------------------------------|-------|-------|-------|-------|-------|-------|-------|
| ...ammar-0.03/lib/Grammar.pm | 100.0 | n/a | n/a | 100.0 | 0.0 | 75.3 | 97.2 |
| blib/lib/PL/FirstFollow.pm | 100.0 | 92.9 | 50.0 | 100.0 | 0.0 | 24.7 | 95.1 |
| Total | 100.0 | 92.9 | 50.0 | 100.0 | 0.0 | 100.0 | 95.5 |
| ----- | ----- | ----- | ----- | ----- | ----- | ----- | ----- |

Si observa un fallo en `PL::FirstFollow` háganoslo saber y además de resolverlo escriba una prueba para detectar el fallo.

Haga un estudio de profiling de su aplicación.

Uso de Templates Un módulo que puede facilitar la escritura de esta práctica es `Text::Template` debido a Mark Jason Dominus. El siguiente ejemplo de uso es un fragmento de un traductor - que nunca acabo de terminar - que toma con fuente un fichero en el formato que usa Moodle para los cuestionarios (conocido como formato GIFT) y lo convierte en un cuestionario \LaTeX :

```
lhp@nereida:~/projects/Gift2LaTeX/Gift2LaTeX/lib$ cat -n Gift2LaTeX.pm
 1  package Gift2LaTeX;
 2
 3  use strict;
 4  use warnings;
 5  use Gift;
 6  use Text::Template;
 7  use HTML::Latex;
..
49  package Gift::TRUEFALSE; # True-false questions belong to this class
50
51  { # closure
52
53      die "Can't find $TEMPLATE_DIR/TRUEFALSE_question.tep\n"
54          unless -e "$TEMPLATE_DIR/TRUEFALSE_question.tep";
55      my $tfq_tmpl = Text::Template->new( #tfq = true-false question
56          DELIMITERS => ['%<', '%>'];
57          SOURCE => "$TEMPLATE_DIR/TRUEFALSE_question.tep",
58      );
..
67  sub gen_latex {
68      my $self = shift;
69
70      ##### Generate latex for question
71      my $prefix = $self->PREFIX;
72
73      my $sufix = $self->POSTSTATE;
74
75      $self->Error("Only HTML and PLAIN formats are supported\n")
76          unless (!$self->FORMAT or ($self->FORMAT =~ m{html|plain}i));
77
78      my ($prefix_tex, $sufix_tex);
79      if (defined($self->FORMAT) and $self->FORMAT =~ m{plain}i) {
80          $prefix_tex = $prefix;
81          $sufix_tex = $sufix;
82      }
83      else { # HTML by default
..
86      }
87      my $params = {
```

```

88     prefix => $prefix_tex,
89     sufix  => $sufix_tex,
90     separator => $separator,
91     label  => $label_prefix.$question_number,
92     question_number => $question_number
93 };
94 my $question_tex = $tfq_tmpl->fill_in(HASH => $params);
96 ##### Generate latex for answer
...     .....
101 }
102 }

```

En la línea 55 se crea el template. El template se lee desde el fichero "\$TEMPLATE_DIR/TRUEFALSE_question.tep" cuyo contenido es una mezcla de texto (en este caso texto L^AT_EX y HTML) con código Perl: El código Perl aparece acotado entre los delimitadores '%<' y '%>'.

```

lhp@nereida:~/projects/Gift2LaTeX/Gift2LaTeX/etc/en$ cat -n TRUEFALSE_question.tep

```

```

1  \ begin{latexonly}
2  %<$separator%>
3  \ label{question:%<$label%>}
4  %<$prefix%>
5
6  \ begin{center}
7  \ begin{tabular}{llll}
8      $\ bigcirc$ & TRUE & $\ bigcirc$ & FALSE
9  \ end{tabular}
10
11  \noindent %<$sufix%>
12  \ end{center}
13  \ end{latexonly}
14
15  \ begin{htmlonly}
16  %<$separator%>
17  \ label{question:%<$label%>}
18  %<$prefix%>
19
20  \ begin{center}
21  \ begin{tabular}{llll}
22      \ href{$\bigcirc$}{answer:%<$label%>} & TRUE &
23      \ href{$\bigcirc$}{answer:%<$label%>} & FALSE
24  \ end{tabular}
25
26  \ noindent %<$sufix%>
27  \ end{center}
28  \ end{htmlonly}

```

El template se rellena en las líneas 87-94. En esa llamada se ejecuta el código Perl incrustado en el esqueleto y su resultado se inserta en la posición que ocupa en el texto.

Concatenación y Documentos HERE Cuando concatene sangre adecuadamente las concatenaciones:

```

my $usage = "Usage: $0 <file> [-full] [-o] [-beans]\n"
            . "Options:\n"

```

```

. "    -full  : produce a full dump\n"
. "    -o      : dump in octal\n"
. "    -beans : source is Java\n"
;

```

ponga el punto al principio de la siguiente línea, no al final.

Pero cuando el número de líneas es grande es mejor usar un *here document* o *documento aqui*. Veamos un ejemplo:

```
print <<"EOI";
```

El programa se deberá ejecutar con:

```
$0 numfiles $opciones initialvalue
EOI
```

Para definir un “documento aqui” se escribe la etiqueta entrecomillada y precedida de << y sigue el texto que constituye el *here document* que se delimita por una línea en blanco que empieza por la etiqueta. Al documento aquí se le trata como una cadena de doble comilla si la etiqueta aparece en doble comilla y como de comilla simple si la etiqueta esta entre comillas simples. Observe que el punto y coma se escribe después de la primera aparición de la etiqueta.

Un problema con el uso de los heredoc es que rompen la estructura normal del sangrado:

```

if ($usage_error) {
    warn <<'END_USAGE';
Usage: qdump <file> [-full] [-o] [-beans]
Options:
    -full  : produce a full dump
    -o      : dump in octal
    -beans : source is Java
END_USAGE
}

```

Es mejor que cada heredoc se aísle en una subrutina y se parametrize con las variables que van a ser interpoladas:

```

sub build_usage {
    my ($prog_name, $file_name) = @_;

    return <<"END_USAGE";
Usage: $prog_name $file_name [-full] [-o] [-beans]
Options:
    -full  : produce a full dump
    -o      : dump in octal
    -beans : source is Java
END_USAGE
}

```

que mas tarde puede ser llamado con los valores de interpolación adecuados:

```

if ($usage_error) {
    warn build_usage($PROGRAM_NAME, $requested_file);
}

```

Véase el libro de Conway Perl Best Practices [?] para mas detalles sobre buenas prácticas de programación con heredocs.

Descarga de los Módulos Necesarios

- El módulo `Grammar` : <http://nereida.deioc.ull.es/~pl/perlexamples/Grammar-0.03.tar.gz>
- El módulo `PL::FirstFollow` : <http://nereida.deioc.ull.es/~pl/perlexamples/PL-FirstFollow-0.02.tar.gz>

3.3. Esquemas de Traducción

Definición 3.3.1. *Un esquema de traducción es una gramática independiente del contexto en la cual se han insertado fragmentos de código en las partes derechas de sus reglas de producción. Los fragmentos de código así insertados se denominan acciones semánticas. Dichos fragmentos actúan, calculan y modifican los atributos asociados con los nodos del árbol sintáctico. El orden en que se evalúan los fragmentos es el de un recorrido primero-profundo del árbol de análisis sintáctico.*

Obsérvese que, en general, para poder aplicar un esquema de traducción hay que construir el árbol sintáctico y después aplicar las acciones empujadas en las reglas en el orden de recorrido primero-profundo. Por supuesto, si la gramática es ambigua una frase podría tener dos árboles y la ejecución de las acciones para ellos podría dar lugar a diferentes resultados. Si se quiere evitar la multiplicidad de resultados (interpretaciones semánticas) es necesario precisar de que árbol sintáctico concreto se está hablando.

Por ejemplo, si en la regla $A \rightarrow \alpha\beta$ insertamos un fragmento de código:

$$A \rightarrow \alpha\{action\}\beta$$

La acción $\{action\}$ se ejecutará después de todas las acciones asociadas con el recorrido del subárbol de α y antes que todas las acciones asociadas con el recorrido del subárbol β .

El siguiente esquema de traducción recibe como entrada una expresión en infijo y produce como salida su traducción a postfijo para expresiones aritméticas con sólo restas de números:

$$\begin{aligned} expr \rightarrow expr_1 - NUM & \quad \{ \$expr\{TRA\} = \$expr[1]\{TRA\}." " . \$NUM\{VAL\}." - " \} \\ expr \rightarrow NUM & \quad \{ \$expr\{TRA\} = \$NUM\{VAL\} \} \end{aligned}$$

Las apariciones de variables sintácticas en una regla de producción se indexan como se ve en el ejemplo, para distinguir de que nodo del árbol de análisis estamos hablando. Cuando hablemos del atributo de un nodo utilizaremos una indexación tipo *hash*. Aquí `VAL` es un atributo de los nodos de tipo `NUM` denotando su valor numérico y para accederlo escribiremos `$NUM{VAL}`. Análogamente `$expr{TRA}` denota el atributo “traducción” de los nodos de tipo `expr`.

Ejercicio 3.3.1. *Muestre la secuencia de acciones a la que da lugar el esquema de traducción anterior para la frase `7 -5 -4`.*

En este ejemplo, el cómputo del atributo `$expr{TRA}` depende de los atributos en los nodos hijos, o lo que es lo mismo, depende de los atributos de los símbolos en la parte derecha de la regla de producción. Esto ocurre a menudo y motiva la siguiente definición:

Definición 3.3.2. *Un atributo tal que su valor en un nodo puede ser computado en términos de los atributos de los hijos del nodo se dice que es un atributo sintetizado.*

Ejemplo 3.3.1. *Un ejemplo de atributo heredado es el tipo de las variables en las declaraciones:*

$$\begin{aligned} decl & \rightarrow type \{ \$list\{T\} = \$type\{T\} \} list \\ type & \rightarrow INT \{ \$type\{T\} = \$int \} \\ type & \rightarrow STRING \{ \$type\{T\} = \$string \} \\ list & \rightarrow ID , \{ \$ID\{T\} = \$list\{T\}; \$list_1\{T\} = \$list\{T\} \} list_1 \\ list & \rightarrow ID \{ \$ID\{T\} = \$list\{T\} \} \end{aligned}$$

Definición 3.3.3. *Un atributo heredado es aquel cuyo valor se computa a partir de los valores de sus hermanos y de su padre.*

Ejercicio 3.3.2. *Escriba un esquema de traducción que convierta expresiones en infijo con los operadores $+-*/()$ y números en expresiones en postfijo. Explique el significado de los atributos elegidos.*

3.4. Recursión por la Izquierda

Definición 3.4.1. *Una gramática es recursiva por la izquierda cuando existe una derivación $A \xRightarrow{*} A\alpha$.*

En particular, es recursiva por la izquierda si contiene una regla de producción de la forma $A \rightarrow A\alpha$. En este caso se dice que la recursión por la izquierda es directa.

Cuando la gramática es *recursiva por la izquierda*, el método de análisis recursivo descendente predictivo no funciona. En ese caso, el procedimiento A asociado con A ciclaría para siempre sin llegar a consumir ningún terminal.

3.5. Introducción

In computer science, a *parsing expression grammar*, or *PEG*, is a type of analytic formal grammar, i.e. it describes a formal language in terms of a set of rules for recognizing strings in the language.

The formalism was introduced by Bryan Ford in 2004 and is closely related to the family of top-down parsing languages introduced in the early 1970s.

Syntactically, PEGs also look similar to context-free grammars (CFGs), but they have a different interpretation:

- the choice operator selects the first match in PEG, while it is ambiguous in CFG.
- This is closer to how string recognition tends to be done in practice, e.g. by a recursive descent parser.

Unlike CFGs, PEGs cannot be ambiguous; *if a string parses, it has exactly one valid parse tree.*

It is conjectured that there exist context-free languages that cannot be parsed by a PEG, but this is not yet proven.

3.5.1. Syntax

Formally, a parsing expression grammar consists of:

- A finite set N of nonterminal symbols.
- A finite set Σ of terminal symbols that is disjoint from N .
- A finite set P of parsing rules.
- An expression e_S termed the starting expression.

Each parsing rule in P has the form $A \leftarrow e$, where A is a nonterminal symbol and e is a *parsing expression*.

A parsing expression is a hierarchical expression similar to a regular expression, which is constructed in the following fashion:

1. An atomic parsing expression consists of:
 - a) any terminal symbol,
 - b) any nonterminal symbol, or
 - c) the empty string ϵ .

2. Given any existing parsing expressions e , e_1 , and e_2 , a new parsing expression can be constructed using the following operators:

- a) Sequence: e_1e_2
- b) Ordered choice: e_1/e_2
- c) Zero-or-more: e^*
- d) One-or-more: e^+
- e) Optional: $e^?$
- f) And-predicate: $\&e$
- g) Not-predicate: $!e$

3.5.2. Semantics

The fundamental difference between context-free grammars and parsing expression grammars is that the PEG's choice operator is **ordered**:

1. If the first alternative succeeds, the second alternative is ignored.
2. Thus ordered choice is not commutative, unlike unordered choice as in context-free grammars.
3. The consequence is that if a CFG is transliterated directly to a PEG, any ambiguity in the former is resolved by deterministically picking one parse tree from the possible parses.
4. By carefully choosing the order in which the grammar alternatives are specified, a programmer has a great deal of control over which parse tree is selected.
5. PEGs can **look ahead** into the input string without actually consuming it
6. The and-predicate expression $\&e$ invokes the sub-expression e , and then succeeds if e succeeds and fails if e fails, *but in either case never consumes any input*.
7. The not-predicate expression $!e$ succeeds if e fails and fails if e succeeds, *again consuming no input in either case*.

3.5.3. Implementing parsers from parsing expression grammars

Any parsing expression grammar can be converted directly into a *recursive descent parser*.

Due to the unlimited lookahead capability that the grammar formalism provides, however, the resulting parser **could exhibit exponential time performance in the worst case**.

It is possible to obtain better performance for any parsing expression grammar by converting its recursive descent parser into **a packrat parser, which always runs in linear time**, at the cost of substantially greater storage space requirements.

*A packrat parser is a form of parser similar to a recursive descent parser in construction, except that during the parsing process **it memoizes** the intermediate results of all invocations of the mutually recursive parsing functions, ensuring that each parsing function is only invoked at most once at a given input position.*

Because of this memoization, a packrat parser has the ability to parse many context-free grammars and any parsing expression grammar (including some that do not represent context-free languages) in linear time.

Examples of memoized recursive descent parsers are known from at least as early as 1993.

Note that this analysis of the performance of a packrat parser **assumes that enough memory is available to hold all of the memoized results**; in practice, if there were not enough memory, some parsing functions might have to be invoked more than once at the same input position, and consequently the parser could take more than linear time.

It is also possible to build LL parsers and LR parsers from parsing expression grammars, with better worst-case performance than a recursive descent parser, but the unlimited lookahead capability of the grammar formalism is then lost. Therefore, not all languages that can be expressed using parsing expression grammars can be parsed by LL or LR parsers.

3.5.4. Lexical Analysis

Parsers for languages expressed as a CFG, such as LR parsers, require a separate tokenization step to be done first, which breaks up the input based on the location of spaces, punctuation, etc.

The tokenization is necessary because of the way these parsers use lookahead to parse CFGs that meet certain requirements in linear time.

PEGs do not require tokenization to be a separate step, and tokenization rules can be written in the same way as any other grammar rule.

3.5.5. Left recursion

PEGs cannot express left-recursive rules where a rule refers to itself without moving forward in the string. For example, the following left-recursive CFG rule:

```
string-of-a -> string-of-a 'a' | 'a'
```

can be rewritten in a PEG using the plus operator:

```
string-of-a <- 'a'+
```

The process of rewriting indirectly left-recursive rules is complex in some packrat parsers, especially when semantic actions are involved.

3.5.6. Referencias y Documentación

- Véase Parsing Expression Grammar
- PEG.js documentation
- Testing PEG.js Online
- Michael's Blog: JavaScript Parser Generators. The PEG.js Tutorial
- The Packrat Parsing and Parsing Expression Grammars Page
- PL101: Create Your Own Programming Language. Véanse [3] y [4]
- PL101: Create Your Own Programming Language: Parsing

3.6. Un Ejemplo Sencillo

Donde

```
[~/srcPLgrado/pegjs/examples(master)]$ pwd -P
/Users/casiano/local/src/javascript/PLgrado/pegjs/examples
[~/srcPLgrado/pegjs/examples(master)]$ git remote -v
dmajda https://github.com/dmajda/pegjs.git (fetch)
dmajda https://github.com/dmajda/pegjs.git (push)
origin git@github.com:crguezl/pegjs.git (fetch)
origin git@github.com:crguezl/pegjs.git (push)

https://github.com/crguezl/pegjs/blob/master/examples/arithmetics.pegjs
```

arithmetics.pegjs

```
[~/Dropbox/src/javascript/PLgrado/pegjs/examples(master)]$ cat arithmetics.pegjs
/*
 * Classic example grammar, which recognizes simple arithmetic expressions like
 * "2*(3+4)". The parser generated from this grammar then computes their value.
 */

start
  = additive

additive
  = left:multiplicative PLUS right:additive { return left + right; }
  / left:multiplicative MINUS right:additive { return left - right; }
  / multiplicative

multiplicative
  = left:primary MULT right:multiplicative { return left * right; }
  / left:primary DIV right:multiplicative { return left / right; }
  / primary

primary
  = integer
  / LEFTPAR additive:additive RIGHTPAR { return additive; }

integer "integer"
  = NUMBER

_ = $[ \t\n\r]*

PLUS = _"+"_
MINUS = _"-"_
MULT = _"*"_
DIV = _"/"_
LEFTPAR = _"("_
RIGHTPAR = _")"_
NUMBER = _ digits:[0-9]+ _ { return parseInt(digits, 10); }
```

main.js

```
[~/Dropbox/src/javascript/PLgrado/pegjs/examples(master)]$ cat main.js
var PEG = require("../arithmetics.js");
var r = PEG.parse("(2+9-1)/2");
console.log(r);
```

Rakefile

```
[~/Dropbox/src/javascript/PLgrado/pegjs/examples(master)]$ cat Rakefile
PEGJS = "../bin/pegjs"
task :default => :run

desc "Compile arithmetics.pegjs"
task :compile do
  sh "#{PEGJS} arithmetics.pegjs"
```



```

end

desc "Run and use the parser generated from arithmetics.pegjs"
task :run => :compile do
  sh "node main.js"
end

```

Compilación

```

[~/Dropbox/src/javascript/PLgrado/pegjs/examples(master)]$ rake
../bin/pegjs arithmetics.pegjs
node main.js
5

```

3.6.1. Asociación Incorrecta para la Resta y la División

Definición 3.6.1. Una gramática es recursiva por la izquierda cuando existe una derivación $A \xRightarrow{*} A\alpha$.

En particular, es recursiva por la izquierda si contiene una regla de producción de la forma $A \rightarrow A\alpha$. En este caso se dice que la recursión por la izquierda es directa.

Cuando la gramática es *recursiva por la izquierda*, el método de análisis recursivo descendente predictivo no funciona. En ese caso, el procedimiento A asociado con A ciclaría para siempre sin llegar a consumir ningún terminal.

Es por eso que hemos escrito las reglas de la calculadora con recursividad a derechas,

```

additive
= left:multiplicative PLUS right:additive { return left + right; }
/ left:multiplicative MINUS right:additive { return left - right; }
/ multiplicative

multiplicative
= left:primary MULT right:multiplicative { return left * right; }
/ left:primary DIV right:multiplicative { return left / right; }
/ primary

```

pero eso da lugar a árboles hundidos hacia la derecha y a una aplicación de las reglas semánticas errónea:

```

[~/pegjs/examples(master)]$ cat main.js
var PEG = require("../arithmetics.js");
var r = PEG.parse("5-3-2");
console.log(r);

[~/pegjs/examples(master)]$ node main.js
4

```

3.7. PegJS en los Browser

La opción -e de pegjs

```

[~/Dropbox/src/javascript/PLgrado/jison]$ pegjs --help
Usage: pegjs [options] [--] [<input_file>] [<output_file>]

```

Generates a parser from the PEG grammar specified in the <input_file> and writes it to the <output_file>.

If the <output_file> is omitted, its name is generated by changing the <input_file> extension to ".js". If both <input_file> and <output_file> are omitted, standard input and output are used.

Options:

| | |
|-----------------------------|---|
| -e, --export-var <variable> | name of the variable where the parser object will be stored (default: "module.exports") |
| --cache | make generated parser cache results |
| --track-line-and-column | make generated parser track line and column |
| -v, --version | print version information and exit |
| -h, --help | print help and exit |

```
[~/Dropbox/src/javascript/PLgrado/pegjs/examples(master)]$ rake web
../bin/pegjs -e calculator arithmetics.pegjs
```

arithmetic.pegjs

```
[~/Dropbox/src/javascript/PLgrado/pegjs/examples(master)]$ cat arithmetics.pegjs
/*
```

```
 * Classic example grammar, which recognizes simple arithmetic expressions like
 * "2*(3+4)". The parser generated from this grammar then computes their value.
 */
```

start

= additive

additive

= left:multiplicative PLUS right:additive { return left + right; }
/ left:multiplicative MINUS right:additive { return left - right; }
/ multiplicative

multiplicative

= left:primary MULT right:multiplicative { return left * right; }
/ left:primary DIV right:multiplicative { return left / right; }
/ primary

primary

= integer
/ LEFTPAR additive:additive RIGHTPAR { return additive; }

integer "integer"

= NUMBER

_ = \$[\t\n\r]*

PLUS = _"+"_

MINUS = _"-"_

MULT = _"*"_

DIV = _"/"_

LEFTPAR = _"("_

RIGHTPAR = _")"_

NUMBER = _ digits:[0-9]+ _ { return parseInt(digits, 10); }

calculator.html

```
[~/Dropbox/src/javascript/PLgrado/pegjs/examples(master)]$ cat calculator.html
```

```
<!DOCTYPE HTML>
```

```
<html lang="en">
```

```
  <head>
```

```
    <meta charset="utf-8">
```

```
    <title>pegjs</title>
```

```
    <link rel="stylesheet" href="global.css" type="text/css" media="screen" charset="utf-8" />
```

```
  </head>
```

```
  <body>
```

```
    <h1>pegjs</h1>
```

```
    <div id="content">
```

```
      <script src="jquery/jquery.js"></script>
```

```
      <script src="arithmetics.js"></script>
```

```
      <script src="calculator.js"></script>
```

```
      <p>
```

```
        Load an example:
```

```
        <input type="file" id="examples" />
```

```
      </p>
```

```
      <p>
```

```
        <table>
```

```
          <tr>
```

```
            <td>
```

```
              <textarea id="input" autofocus cols = "40" rows = "4">2+3*4</textarea>
```

```
            </td>
```

```
            <td class="output">
```

```
              <pre>
```

```
<span id="output">
```

```
</span> <!-- Output goes here! -->
```

```
              </pre>
```

```
            </td>
```

```
            <td><button id="eval" type="button">eval</button></td>
```

```
          </tr>
```

```
        </table>
```

```
      </p>
```

```
    </div>
```

```
  </body>
```

```
</html>
```

calculator.js

```
[~/Dropbox/src/javascript/PLgrado/pegjs/examples(master)]$ cat calculator.js
```

```
$(document).ready(function() {
```

```
  $('#eval').click(function() {
```

```
    try {
```

```
      var result = calculator.parse($('#input').val());
```

```
      $('#output').html(result);
```

```
    } catch (e) {
```

```
      $('#output').html('<div class="error"><pre>\n' + String(e) + '\n</pre></div>');
```



Figura 3.1: pegjs en la web

```

    }
  });

$("#examples").change(function(ev) {
  var f = ev.target.files[0];
  var r = new FileReader();
  r.onload = function(e) {
    var contents = e.target.result;

    input.innerHTML = contents;
  }
  r.readAsText(f);
});

});

```

3.8. Eliminación de la Recursividad por la Izquierda

Es posible modificar la gramática para eliminar la recursión por la izquierda. En este apartado nos limitaremos al caso de recursión por la izquierda directa. La generalización al caso de recursión por la izquierda no-directa se reduce a la iteración de la solución propuesta para el caso directo.

Consideremos una variable A con dos producciones:

$$A \rightarrow A\alpha \mid \beta$$

donde $\alpha, \beta \in (V \cup \Sigma)^*$ no comienzan por A . Estas dos producciones pueden ser sustituidas por:

$$A \rightarrow \beta\alpha^*$$

eliminando así la recursión por la izquierda.

3.8.1. Eliminación de la Recursividad por la izquierda Respetando el Orden de Ejecución de las Acciones Semánticas

```

[~/pegjs-coffee-remove-left(master)]$ cat -n remove_left_recursive.pegjs
1  /*
2

```

```

3 Exercise: Find a PEG equivalent to the following left-recursive
4 grammar:
5
6 A : A 'x' { $$ = do_x($1, $2); } | 'y' { $$ = do_y($1); }
7
8 */
9
10 {
11     @do_y = (y)    -> console.log("do_y({y})"); y
12     @do_x = (a, x)-> console.log("do_x({a}, {x})"); a+x
13 }
14
15 A = y:'y' xs:('x'*)
16     {
17         a = @do_y(y)
18         for x in xs
19             a = @do_x(a, x)
20         a
21     }

```

```

[~/pegjs-coffee-remove-left(master)]$ pegcoffee remove_left_recursive.pegjs
[~/pegjs-coffee-remove-left(master)]$ ls -ltr | tail -1
-rw-rw-r-- 1 casiano staff 8919 3 jun 10:42 remove_left_recursive.js

```

```

[~/pegjs-coffee-remove-left(master)]$ cat use_remove_left.coffee
PEG = require("./remove_left_recursive.js")
inputs = [
    "yxx"
    "y"
    "yxxx"
]

```

```

for input in inputs
    console.log("input = #{input}")
    r = PEG.parse input
    console.log("result = #{r}\n")

```

```

[~/pegjs-coffee-remove-left(master)]$ coffee use_remove_left.coffee
input = yxx
do_y(y)
do_x(y, x)
do_x(yx, x)
result = yxx

```

```

input = y
do_y(y)
result = y

```

```

input = yxxx
do_y(y)
do_x(y, x)
do_x(yx, x)
do_x(yxx, x)
result = yxxx

```

3.8.2. Eliminando la Recursividad por la Izquierda en la Calculadora

```
[~/Dropbox/src/javascript/PLgrado/pegjs/examples(master)]$ cat simple.pegjs
/* From the Wikipedia
Value  ← [0-9]+ / '(' Expr ')'
Product ← Value (('*' / '/') Value)*
Sum     ← Product (('+' / '-') Product)*
Expr    ← Sum
*/
{
  function reduce(left, right) {
    var sum = left;
    // console.log("sum = "+sum);
    for(var i = 0; i < right.length;i++) {
      var t = right[i];
      var op = t[0];
      var num = t[1];
      switch(op) {
        case '+' : sum += num; break;
        case '-' : sum -= num; break;
        case '*' : sum *= num; break;
        case '/' : sum /= num; break;
        default : console.log("Error! "+op);
      }
      // console.log("sum = "+sum);
    }
    return sum;
  }
}

sum  = left:product right:($[+-] product)* { return reduce(left, right); }
product = left:value right:($[*/] value)*   { return reduce(left, right); }
value  = number:$[0-9]+                      { return parseInt(number,10); }
      / '(' sum:sum ')'                      { return sum; }
```

Es posible especificar mediante llaves un código que este disponible dentro de las acciones semánticas.

Ejecución:

```
[~/pegjs/examples(master)]$ cat use_simple.js
var PEG = require("./simple.js");
var r = PEG.parse("2-3-4");
console.log(r);
```

```
[~/pegjs/examples(master)]$ node use_simple.js
-5
```

Veamos otra ejecución:

```
[~/Dropbox/src/javascript/PLgrado/pegjs/examples(master)]$ cat use_simple.js
var PEG = require("./simple.js");
var r = PEG.parse("2+3*(2+1)-10/2");
console.log(r);
```

```
[~/Dropbox/src/javascript/PLgrado/pegjs/examples(master)]$ ../bin/pegjs simple.pegjs
```

```
[~/Dropbox/src/javascript/PLgrado/pegjs/examples(master)]$ node use_simple.js
```

6

3.9. Eliminación de la Recursividad por la Izquierda y Atributos Heredados

La sección anterior da una forma sencilla de resolver el problema respetando la semántica. Si no se dispone de operadores de repetición la cosa se vuelve mas complicada. Las siguientes secciones muestran una solución para transformar un esquema de traducción recursivo por la izquierda en otro no recursivo por la izquierda respetando el orden en el que se ejecutan las acciones semánticas. Por último se ilustra como se puede aplicar esta técnica en `pegjs` (aunque obviamente es mucho mejor usar la ilustrada anteriormente).

3.9.1. Eliminación de la Recursión por la Izquierda en la Gramática

Es posible modificar la gramática para eliminar la recursión por la izquierda. En este apartado nos limitaremos al caso de recursión por la izquierda directa. La generalización al caso de recursión por la izquierda no-directa se reduce a la iteración de la solución propuesta para el caso directo.

Consideremos una variable A con dos producciones:

$$A \rightarrow A\alpha \mid \beta$$

donde $\alpha, \beta \in (V \cup \Sigma)^*$ no comienzan por A . Estas dos producciones pueden ser sustituidas por:

$$\begin{aligned} A &\rightarrow \beta R \\ R &\rightarrow \alpha R \mid \epsilon \end{aligned}$$

eliminando así la recursión por la izquierda.

Definición 3.9.1. *La producción $R \rightarrow \alpha R$ se dice recursiva por la derecha.*

Las producciones recursivas por la derecha dan lugar a árboles que se hunden hacia la derecha. Es mas difícil traducir desde esta clase de árboles operadores como el menos, que son asociativos a izquierdas.

Ejercicio 3.9.1. *Elimine la recursión por la izquierda de la gramática*

$$\begin{aligned} expr &\rightarrow expr - NUM \\ expr &\rightarrow NUM \end{aligned}$$

3.9.2. Eliminación de la Recursión por la Izquierda en un Esquema de Traducción

La eliminación de la recursión por la izquierda es sólo un paso: debe ser extendida a esquemas de traducción, de manera que no sólo se preserve el lenguaje sino la secuencia de acciones. Supongamos que tenemos un esquema de traducción de la forma:

$$\begin{aligned} A &\rightarrow A\alpha \quad \{ \text{alpha_action} \} \\ A &\rightarrow A\beta \quad \{ \text{beta_action} \} \\ A &\rightarrow \gamma \quad \{ \text{gamma_action} \} \end{aligned}$$

para una sentencia como $\gamma\beta\alpha$ la secuencia de acciones será:

`gamma_action beta_action alpha_action`

¿Cómo construir un esquema de traducción para la gramática resultante de eliminar la recursión por la izquierda que ejecute las acciones asociadas en el mismo orden?. Supongamos para simplificar, que las acciones no dependen de atributos ni computan atributos, sino que actúan sobre variables globales. En tal caso, la siguiente ubicación de las acciones da lugar a que se ejecuten en el mismo orden:

$$\begin{aligned} A &\rightarrow \gamma \{ \text{gamma_action} \} R \\ R &\rightarrow \beta \{ \text{beta_action} \} R \\ R &\rightarrow \alpha \{ \text{alpha_action} \} R \\ R &\rightarrow \epsilon \end{aligned}$$

Si hay atributos en juego, la estrategia para construir un esquema de traducción equivalente para la gramática resultante de eliminar la recursividad por la izquierda se complica. Consideremos de nuevo el esquema de traducción de infijo a postfijo de expresiones aritméticas de restas:

$$\begin{aligned} \text{expr} &\rightarrow \text{expr}_1 - \text{NUM} & \{ \$\text{expr}\{\text{T}\} = \$\text{expr}[1]\{\text{T}\} \cdot " \cdot \$\text{NUM}\{\text{VAL}\} \cdot " - " \} \\ \text{expr} &\rightarrow \text{NUM} & \{ \$\text{expr}\{\text{T}\} = \$\text{NUM}\{\text{VAL}\} \} \end{aligned}$$

En este caso introducimos un atributo H para los nodos de la clase *r* el cuál acumula la traducción a postfijo hasta el momento. Observe como este atributo se computa en un nodo *r* a partir del correspondiente atributo del el padre y/o de los hermanos del nodo:

$$\begin{aligned} \text{expr} &\rightarrow \text{NUM} \{ \$r\{\text{H}\} = \$\text{NUM}\{\text{VAL}\} \} r \{ \$\text{expr}\{\text{T}\} = \$r\{\text{T}\} \} \\ r &\rightarrow -\text{NUM} \{ \$r_1\{\text{H}\} = \$r\{\text{H}\} \cdot " \cdot \$\text{NUM}\{\text{VAL}\} \cdot " - " \} r_1 \{ \$r\{\text{T}\} = \$r_1\{\text{T}\} \} \\ r &\rightarrow \epsilon \{ \$r\{\text{T}\} = \$r\{\text{H}\} \} \end{aligned}$$

El atributo H es un ejemplo de atributo heredado.

3.9.3. Eliminación de la Recursividad por la Izquierda en PEGJS

PegJS no permite acciones intermedias. Tampoco se puede acceder al atributo de la parte izquierda. Por eso, a la hora de implantar la solución anterior debemos introducir variables sintácticas temporales que produzcan la palabra vacía y que vayan acompañadas de la acción semántica correspondiente.

Además nos obliga a usar variables visibles por todas las reglas semánticas para emular el acceso a los atributos de la parte izquierda de una regla de producción.

El siguiente ejemplo ilustra como eliminar la recursión por la izquierda respetando la asociatividad de la operación de diferencia:

```
[~/pegjs/examples(master)]$ cat inherited.pegjs
{
  var h = 0, number = 0;
}
e = NUMBER aux1 r      { return h; }
aux1 = /* empty */     { h = number; }

r =  '-' NUMBER aux2 r { return h; }
    / /* empty */
aux2 = /* empty */     { h -= number; }

NUMBER = _ digits:$[0-9]+ _ { number = parseInt(digits, 10); return number; }

_ = $[ \t\n\r]*

[~/pegjs/examples(master)]$ cat use_inherited.js
var PEG = require("./inherited.js");
var r = PEG.parse("2-1-1");
```



```

console.log(r);

var r = PEG.parse("4-2-1");
console.log(r);

var r = PEG.parse("2-3-1");
console.log(r);

[~/pegjs/examples(master)]$ pegjs inherited.pegjs
Referenced rule "$" does not exist.
[~/pegjs/examples(master)]$ ../bin/pegjs inherited.pegjs
[~/pegjs/examples(master)]$ node use_inherited.js
0
1
-2

```

3.10. Asociando un else con su if mas cercano

```

$ cat danglingelse.pegjs
/*
S ← 'if' C 'then' S 'else' S / 'if' C 'then' S
*/

S =   if C:C then S1:S else S2:S { return [ 'ifthenelse', C, S1, S2 ]; }
      / if C:C then S:S           { return [ 'ifthen', C, S ]; }
      / 0                         { return '0'; }
_ = ' '*
C = '_c'_                         { return 'c'; }
0 = '_o'_                         { return 'o'; }
else = '_else'_
if = '_if'_
then = '_then'_

$ cat use_danglingelse.js
var PEG = require("../danglingelse.js");
var r = PEG.parse("if c then if c then o else o");
console.log(r);

$ ../bin/pegjs danglingelse.pegjs
$ node use_danglingelse.js
[ 'ifthen', 'c', [ 'ifthenelse', 'c', '0', '0' ] ]

```

3.11. Comentarios Anidados

```

$ cat pascal_comments.pegjs
/* Pascal nested comments */

P      =   prog:N+                { return prog; }
N      =   chars:$(!Begin ANY)+  { return chars;}
        / C
C      =   Begin chars:T* End     { return chars.join(''); }
T      =   C

```

```

        / (!Begin !End char:ANY)          { return char;}
Begin = '(*'
End   = '*)'
ANY   = 'z' /* any character */          { return 'z'; }
        / char:[^z]                       { return char; }

$ cat use_pascal_comments.js
var PEG = require("./pascal_comments.js");
var r = PEG.parse(
  "not bla bla (* pascal (* nested *) comment *)"+
  " pum pum (* another comment *)");
console.log(r);

$ ../bin/pegjs pascal_comments.pegjs
$ node use_pascal_comments.js
[ 'not bla bla ',
  ' pascal nested comment ',
  ' pum pum ',
  ' another comment ' ]

```

3.12. Un Lenguaje Dependiente del Contexto

El lenguaje $\{a^n b^n c^n / n \in \mathcal{N}\}$ no puede ser expresado mediante una gramática independiente del contexto.

```

[~/Dropbox/src/javascript/PLgrado/pegjs/examples(master)]$ cat anbncn.pegjs
/*
  The following parsing expression grammar describes the classic
  non-context-free language :
      { anbncn / n >= 1 }

      S ← &(A 'c') 'a'+ B !('a'/'b'/'c')
      A ← 'a' A? 'b'
      B ← 'b' B? 'c'

*/

S = &(A 'c') 'a'+ B !('a'/'b'/'c')
A = 'a' A? 'b'
B = 'b' B? 'c'

```

Este ejemplo puede ser obtenido desde GitHub:

```

[~/Dropbox/src/javascript/PLgrado/pegjs/examples(master)]$ git remote -v
dmajda https://github.com/dmajda/pegjs.git (fetch)
dmajda https://github.com/dmajda/pegjs.git (push)
origin git@github.com:crguezl/pegjs.git (fetch)
origin git@github.com:crguezl/pegjs.git (push)

```

Veamos un ejemplo de uso:

```

[~/Dropbox/src/javascript/PLgrado/pegjs/examples(master)]$ cat use_anbncn.js
var PEG = require("./anbncn.js");
var r = PEG.parse("aabbcc");
console.log(r);

```

```

try {
  r = PEG.parse("aabbc");
  console.log(r);
}
catch (e) {
  console.log("Grr...."+e);
}

```

Ejecución:

```

[~/Dropbox/src/javascript/PLgrado/pegjs/examples(master)]$ ../bin/pegjs anbncn.pegjs

[~/Dropbox/src/javascript/PLgrado/pegjs/examples(master)]$ node use_anbncn.js
[ '', [ 'a', 'a' ], [ 'b', [ 'b', '', 'c' ], 'c' ], '' ]
Grr....SyntaxError: Expected "c" but end of input found.

```

3.13. Usando Pegjs con CoffeeScript

Instalación de pegjs-coffee-plugin

```

[~/Dropbox/src/javascript/PLgrado/pegjs/examples(master)]$ sudo npm install -g pegjs-coffee-p

```

Ejemplo Sencillo

```

[~/Dropbox/src/javascript/PLgrado/pegjs-coffee-plugin/examples(master)]$ cat simple.pegjs
{
  @reduce = (left, right)->
    sum = left
    for t in right
      op = t[0]
      num = t[1]
      switch op
        when '+' then sum += num; break
        when '-' then sum -= num; break
        when '*' then sum *= num; break
        when '/' then sum /= num; break
      else console.log("Error! "+op)
    sum

  sum = left:product right:([+-] product)* { @reduce(left, right); }
  product = left:value right:([*/] value)* { @reduce(left, right); }
  value = number:[0-9]+ { parseInt(number.join(''),10) }
         / '(' sum:sum ')' { sum }
}

```

```

[~/Dropbox/src/javascript/PLgrado/pegjs-coffee-plugin/examples(master)]$ cat use_simple.coffee
PEG = require("./simple.js")
r = PEG.parse("2+3*(2+1)-10/2")
console.log(r)

```

```

[~/Dropbox/src/javascript/PLgrado/pegjs-coffee-plugin/examples(master)]$ cat Rakefile

task :default do
  sh "pegcoffee simple.pegjs"

```

```
end
```

```
task :run do
  sh "coffee use_simple.coffee"
end
```

```
[~/Dropbox/src/javascript/PLgrado/pegjs-coffee-plugin/examples(master)]$ rake
pegcoffee simple.pegjs
[~/Dropbox/src/javascript/PLgrado/pegjs-coffee-plugin/examples(master)]$ rake run
coffee use_simple.coffee
6
```

Véase También

- pegjs-coffee-plugin en GitHub

3.14. Práctica: Ambigüedad en C++

This lab illustrates a problem that arises in C++. The C++ syntax does not disambiguate between expression statements (**stmt**) and declaration statements (**decl**). The ambiguity arises when an expression statement has a function-style cast as its left-most subexpression. Since C does not support function-style casts, this ambiguity does not occur in C programs. For example, the phrase

```
int (x) = y+z;
```

parses as either a **decl** or a **stmt**.

The disambiguation rule used in C++ is that *if the statement can be interpreted both as a declaration and as an expression, the statement is interpreted as a declaration statement*.

The following examples disambiguate into *expression* statements when the potential *declarator* is followed by an operator different from equal or semicolon (**type_spec** stands for a type specifier):

expr	dec
type_spec(i)++;	type_spec(*i)(int);
type_spec(i,3)<<d;	type_spec(j)[5];
type_spec(i)->l=24;	type_spec(m) = { 1, 2 };
	type_spec(a);
	type_spec(*b)();
	type_spec(c)=23;
	type_spec(d),e,f,g=0;
	type_spec(h)(e,3);

Regarding to this problem, Bjarne Stroustrup remarks:

Consider analyzing a statement consisting of a sequence of tokens as follows:

type_spec (dec_or_exp) tail

*Here **dec_or_exp** must be a declarator, an expression, or both for the statement to be legal. This implies that **tail** must be a semicolon, something that can follow a parenthesized declarator or something that can follow a parenthesized expression, that is, an initializer, **const**, **volatile**, **(**, **[**, or a postfix or infix operator. The general cases cannot be resolved without backtracking, nested grammars or similar advanced parsing strategies. In particular, the lookahead needed to disambiguate this case is not limited.*

The following grammar depicts an oversimplified version of the C++ ambiguity:

```

$ cat CplusplusNested.y
%token ID INT NUM

%right '='
%left '+'

%%
prog:
    /* empty */
    | prog stmt
    ;

stmt:
    expr ';'
    | decl
    ;

expr:
    ID
    | NUM
    | INT '(' expr ')' /* typecast */
    | expr '+' expr
    | expr '=' expr
    ;

decl:
    INT declarator ';'
    | INT declarator '=' expr ';'
    ;

declarator:
    ID
    | '(' declarator ')'
    ;

%%

```

Escriba un programa PegJS en CoffeeScript que distinga correctamente entre declaraciones y sentencias. Este es un ejemplo de un programa que usa una solución al problema:

```

[~/Dropbox/src/javascript/PLgrado/pegjs-coffee-plugin/examples(master)]$ cat use_cplusplus.coffee
PEG = require("./cplusplus.js")
input = "int (a); int c = int (b);"

r = PEG.parse(input)
console.log("input = '#{input}'\noutput="+JSON.stringify r)

input = "int b = 4+2 ; "
r = PEG.parse(input)
console.log("input = '#{input}'\noutput="+JSON.stringify r)

input = "bum = caf = 4-1;\n"
r = PEG.parse(input)
console.log("input = '#{input}'\noutput="+JSON.stringify r)

```

```

input = "b2 = int(4);"
r = PEG.parse(input)
console.log("input = '#{input}'\noutput="+JSON.stringify r)

input = "int(4);"
r = PEG.parse(input)
console.log("input = '#{input}'\noutput="+JSON.stringify r)

```

Y este un ejemplo de salida:

```

$ pegcoffee cplusplus.pegjs
$ coffee use_cplusplus.coffee
input = 'int (a); int c = int (b);'
output=["decl","decl"]
input = 'int b = 4+2 ; '
output=["decl"]
input = 'bum = caf = 4-1;
'
output=["stmt"]
input = 'b2 = int(4);'
output=["stmt"]
input = 'int(4);'
output=["stmt"]

```

3.15. Práctica: Inventando un Lenguaje: Tortoise

El objetivo de esta práctica es crear un lenguaje de programación imperativa sencillo de estilo LOGO. Para ello lea el capítulo *Inventing a Language - Tortoise* del curso PL101: Create Your Own Programming de Nathan Whitehead. Haga todos los ejercicios e implemente el lenguaje descrito.

Puede encontrar una solución a la práctica en GitHub en el repositorio pl101 de Dave Ingram. Úsela como guía cuando se sienta desorientado.

Recursos

- *Inventing a Language - Tortoise* por Nathan Whitehead
- Repositorio dingram / pl101 en GitHub con las soluciones a esta práctica.
 - Blog de dingram (Dave Ingram)
- Repositorio PatricxCR / PL101 en GitHub con las soluciones a esta práctica.
- Repositorio Clinton N. Dreisbach / PL101 en GitHub con contenidos del curso PL101
- Foro
- Sobre Nathan Whitehead
 - Nathan's Lessons
 - Nathan Whitehead en GitHub
 - Nathan in YouTube

Capítulo 4

Análisis Sintáctico Ascendente en JavaScript

4.1. Conceptos Básicos para el Análisis Sintáctico

Suponemos que el lector de esta sección ha realizado con éxito un curso en teoría de autómatas y lenguajes formales. Las siguientes definiciones repasan los conceptos mas importantes.

Definición 4.1.1. Dado un conjunto A , se define A^* el cierre de Kleene de A como: $A^* = \bigcup_{n=0}^{\infty} A^n$. Se admite que $A^0 = \{\epsilon\}$, donde ϵ denota la palabra vacía, esto es la palabra que tiene longitud cero, formada por cero símbolos del conjunto base A .

Definición 4.1.2. Una gramática G es una cuaterna $G = (\Sigma, V, P, S)$. Σ es el conjunto de terminales. V es un conjunto (disjunto de Σ) que se denomina conjunto de variables sintácticas o categorías gramaticales, P es un conjunto de pares de $V \times (V \cup \Sigma)^*$. En vez de escribir un par usando la notación $(A, \alpha) \in P$ se escribe $A \rightarrow \alpha$. Un elemento de P se denomina producción. Por último, S es un símbolo del conjunto V que se denomina símbolo de arranque.

Definición 4.1.3. Dada una gramática $G = (\Sigma, V, P, S)$ y $\mu = \alpha A \beta \in (V \cup \Sigma)^*$ una frase formada por variables y terminales y $A \rightarrow \gamma$ una producción de P , decimos que μ deriva en un paso en $\alpha \gamma \beta$. Esto es, derivar una cadena $\alpha A \beta$ es sustituir una variable sintáctica A de V por la parte derecha γ de una de sus reglas de producción. Se dice que μ deriva en n pasos en δ si deriva en $n - 1$ pasos en una cadena $\alpha A \beta$ la cual deriva en un paso en δ . Se escribe entonces que $\mu \xRightarrow{*} \delta$. Una cadena deriva en 0 pasos en si misma.

Definición 4.1.4. Dada una gramática $G = (\Sigma, V, P, S)$ se denota por $L(G)$ o lenguaje generado por G al lenguaje:

$$L(G) = \{x \in \Sigma^* : S \xRightarrow{*} x\}$$

Esto es, el lenguaje generado por la gramática G esta formado por las cadenas de terminales que pueden ser derivados desde el símbolo de arranque.

Definición 4.1.5. Una derivación que comienza en el símbolo de arranque y termina en una secuencia formada por sólo terminales de Σ se dice completa.

Una derivación $\mu \xRightarrow{*} \delta$ en la cual en cada paso $\alpha A x$ la regla de producción aplicada $A \rightarrow \gamma$ se aplica en la variable sintáctica mas a la derecha se dice una derivación a derechas

Una derivación $\mu \xRightarrow{*} \delta$ en la cual en cada paso $x A \alpha$ la regla de producción aplicada $A \rightarrow \gamma$ se aplica en la variable sintáctica mas a la izquierda se dice una derivación a izquierdas

Definición 4.1.6. Observe que una derivación puede ser representada como un árbol cuyos nodos están etiquetados en $V \cup \Sigma$. La aplicación de la regla de producción $A \rightarrow \gamma$ se traduce en asignar como hijos del nodo etiquetado con A a los nodos etiquetados con los símbolos $X_1 \dots X_n$ que constituyen la frase $\gamma = X_1 \dots X_n$. Este árbol se llama árbol sintáctico concreto asociado con la derivación.

Definición 4.1.7. Observe que, dada una frase $x \in L(G)$ una derivación desde el símbolo de arranque da lugar a un árbol. Ese árbol tiene como raíz el símbolo de arranque y como hojas los terminales $x_1 \dots x_n$ que forman x . Dicho árbol se denomina árbol de análisis sintáctico concreto de x . Una derivación determina una forma de recorrido del árbol de análisis sintáctico concreto.

Definición 4.1.8. Una gramática G se dice ambigua si existe alguna frase $x \in L(G)$ con al menos dos árboles sintácticos. Es claro que esta definición es equivalente a afirmar que existe alguna frase $x \in L(G)$ para la cual existen dos derivaciones a izquierda (derecha) distintas.

4.1.1. Ejercicio

Dada la gramática con producciones:

```

program → declarations statements | statements
declarations → declaration ';' declarations | declaration ';'
declaration → INT idlist | STRING idlist
statements → statement ';' statements | statement
statement → ID '=' expression | P expression
expression → term '+' expression | term
term → factor '*' term | factor
factor → '(' expression ')' | ID | NUM | STR
idlist → ID ',' idlist | ID

```

En esta gramática, Σ esta formado por los caracteres entre comillas simples y los símbolos cuyos identificadores están en mayúsculas. Los restantes identificadores corresponden a elementos de V . El símbolo de arranque es $S = \text{program}$.

Conteste a las siguientes cuestiones:

1. Describa con palabras el lenguaje generado.
2. Construya el árbol de análisis sintáctico concreto para cuatro frases del lenguaje.
3. Señale a que recorridos del árbol corresponden las respectivas derivaciones a izquierda y a derecha en el apartado 2.
4. ¿Es ambigua esta gramática?. Justifique su respuesta.

4.2. Ejemplo Simple en Jison

Jison es un generador de analizadores sintácticos LALR. Otro analizador LALR es JS/CC.

Gramática

%%

```

S      : A
      ;
A      : /* empty */
      | A x
      ;

```

basic2_lex.jison

```

[~/jison/examples/basic2_lex(develop)]$ cat basic2_lex.jison
/* description: Basic grammar that contains a nullable A nonterminal. */

```



```

%lex
%%

\s+          { /* skip whitespace */}
[a-zA-Z_]\w* {return 'x';}

/lex

%%

S : A
    { return $1+" identifiers"; }
;
A : /* empty */
    {
        console.log("starting");
        $$ = 0;
    }
| A x {
        $$ = $1 + 1;
        console.log($$)
    }
;

```

index.html

```

$ cat basic2_lex.html
<!DOCTYPE HTML>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Jison</title>
    <link rel="stylesheet" href="global.css" type="text/css" media="screen" charset="utf-8" />
  </head>
  <body>
    <h1>basic2_lex demo</h1>
    <div id="content">
      <script src="jquery/jquery.js"></script>
      <script src="basic2_lex.js"></script>
      <script src="main.js"></script>
      <p>
        <input type="text" value="x x x x" /> <button>parse</button>
        <span id="output"></span> <!-- Output goes here! -->
      </p>
    </div>
  </body>
</html>

```

Rakefile

```

$ cat Rakefile
# install package:
#   sudo npm install beautifier
#

```

```
# more about beautifier:
#      https://github.com/rickeyski/node-beautifier

dec "compile the grammar basic2_lex_ugly.jison"
task :default => %w{basic2_lex_ugly.js} do
  sh "mv basic2_lex.js basic2_lex_ugly.js"
  sh "jsbeautify basic2_lex_ugly.js > basic2_lex.js"
  sh "rm -f basic2_lex_ugly.js"
end

file "basic2_lex_ugly.js" => %w{basic2_lex.jison} do
  sh "jison basic2_lex.jison -o basic2_lex.js"
end
```

1. node-beautifier

Véase También

1. JISON
2. Try Jison Examples
3. JavaScript 1.4 LR(1) Grammar 1999.
4. Creating a JavaScript Parser Una implementación de ECMAScript 5.1 usando Jison disponible en GitHub en <https://github.com/cjihrig/jsparser>. Puede probarse en: <http://www.cjihrig.com/development>
5. Bison on JavaScript por Rolando Perez
6. Slogo a language written using Jison
7. List of languages that compile to JS
8. Prototype of a Scannerless, Generalized Left-to-right Rightmost (SGLR) derivation parser for JavaScript

global.css

```
[~/jison/examples/basic2_lex(develop)]$ cat global.css
html *
{
  font-size: large;
  /* The !important ensures that nothing can override what you've set in this style (unless i
  font-family: Arial;
}

.thumb {
  height: 75px;
  border: 1px solid #000;
  margin: 10px 5px 0 0;
}

h1          { text-align: center; font-size: x-large; }
th, td      { vertical-align: top; text-align: left; }
/* #finaltable * { color: white; background-color: black; } */

/* #finaltable table { border-collapse: collapse; } */
```

```

/* #finaltable table, td { border:1px solid white; } */
#finaltable:hover td { background-color: blue; }
tr:nth-child(odd)    { background-color:#eee; }
tr:nth-child(even)   { background-color:#00FF66; }
input               { text-align: right; border: none;          } /* Align input to the right */
textarea           { border: outset; border-color: white;      }
table              { border: inset; border-color: white; }
.hidden            { display: none; }
.unhidden          { display: block; }
table.center       { margin-left:auto; margin-right:auto; }
#result            { border-color: red; }
tr.error           { background-color: red; }
pre.output         { background-color: white; }
span.repeated      { background-color: red }
span.header        { background-color: blue }
span.comments      { background-color: orange }
span.blanks        { background-color: green }
span.nameEqualValue { background-color: cyan }
span.error         { background-color: red }

body
{
  background-color:#b0c4de; /* blue */
}

```

4.2.1. Véase También

1. JISON
2. Try Jison Examples
3. JavaScript 1.4 LR(1) Grammar 1999.
4. Creating a JavaScript Parser Una implementación de ECAMScript 5.1 usando Jison disponible en GitHub en <https://github.com/cjihrig/jsparser>. Puede probarse en: <http://www.cjihrig.com/development>
5. Slogo a language written using Jison
6. List of languages that compile to JS
7. Prototype of a Scannerless, Generalized Left-to-right Rightmost (SGLR) derivation parser for JavaScript

4.2.2. Práctica: Secuencia de Asignaciones Simples

Modifique este ejemplo para que el lenguaje acepte una secuencia de sentencias de asignación de la forma `ID = NUM` separadas por puntos y comas, por ejemplo `a = 4; b = 4.56; c = -8.57e34`. El analizador retorna un hash/objeto cuyas claves son los identificadores y cuyos valores son los números. Clone el repositorio en <https://github.com/crguezl/jison-basic2>.

Modifique los analizadores léxico y sintáctico de forma conveniente.

Añada acciones semánticas para que el analizador devuelva una tabla de símbolos con los identificadores y sus valores.

4.3. Ejemplo en Jison: Calculadora Simple

1. Enlace al fork del proyecto jison de crguezl (GitHub)

calculator.json

```
[~/jison/examples/html_calc_example(develop)]$ cat calculator.json
```

```
/* description: Parses and executes mathematical expressions. */

/* lexical grammar */
%lex
%%

\s+                /* skip whitespace */
[0-9]+("."[0-9]+)?\b return 'NUMBER'
"*"               return '*'
"/"              return '/'
"_"              return '-'
"+"             return '+'
"^"             return '^'
"!"             return '!'
"%"             return '%'
"("             return '('
")"             return ')'
"PI"            return 'PI'
"E"            return 'E'
<<EOF>>        return 'EOF'
.               return 'INVALID'

/lex

/* operator associations and precedence */

%left '+' '-'
%left '*' '/'
%left '^'
%right '!'
%right '%'
%left UMINUS

%start expressions

%% /* language grammar */

expressions
: e EOF
  { typeof console !== 'undefined' ? console.log($1) : print($1);
    return $1; }
;

e
: e '+' e
  {$$ = $1+$3;}
| e '-' e
  {$$ = $1-$3;}
| e '*' e
  {$$ = $1*$3;}
```

```

| e '/' e
    {$$ = $1/$3;}
| e '^' e
    {$$ = Math.pow($1, $3);}
| e '!'
    {{
        $$ = (function fact (n) { return n==0 ? 1 : fact(n-1) * n } )($1);
    }}
| e '%'
    {$$ = $1/100;}
| '-' e %prec UMINUS
    {$$ = -$2;}
| '(' e ')'
    {$$ = $2;}
| NUMBER
    {$$ = Number(yytext);}
| E
    {$$ = Math.E;}
| PI
    {$$ = Math.PI;}
;

```

main.js

```

[~/jison/examples/html_calc_example(develop)]$ cat main.js
$(document).ready(function () {
    $("button").click(function () {
        try {
            var result = calculator.parse($("#input").val())
            $("#span").html(result);
        } catch (e) {
            $("#span").html(String(e));
        }
    });
});

```

calculator.html

```

[~/jison/examples/html_calc_example(develop)]$ cat calculator.html
<!DOCTYPE HTML>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Calc</title>
    <link rel="stylesheet" href="global.css" type="text/css" media="screen" charset="utf-8" />
  </head>
  <body>
    <h1>Calculator demo</h1>
    <div id="content">
      <script src="jquery/jquery.js"></script>
      <script src="calculator.js"></script>
      <script src="main.js"></script>
      <p>
        <input type="text" value="PI*4^2 + 5" /> <button>equals</button>
      </p>
    </div>
  </body>
</html>

```

```

        <span></span> <!-- Output goes here! -->
    </p>
</div>
</body>
</html>

```

Rakefile

```

[~/jisoncalc(clase)]$ cat Rakefile
task :default => %w{calcugly.js} do
  sh "jsbeautify calcugly.js > calculator.js"
  sh "rm -f calcugly.js"
end

file "calcugly.js" => %w{calculator.jison} do
  sh "jison calculator.jison calculator.l -o calculator.js; mv calculator.js calcugly.js"
end

task :testf do
  sh "open -a firefox test/test.html"
end

task :tests do
  sh "open -a safari test/test.html"
end

```

global.css

```

[~/jison/examples/html_calc_example(develop)]$ cat global.css
html *
{
    font-size: large;
    /* The !important ensures that nothing can override what you've set in this style (unless i
    font-family: Arial;
}

.thumb {
    height: 75px;
    border: 1px solid #000;
    margin: 10px 5px 0 0;
}

h1          { text-align: center; font-size: x-large; }
th, td      { vertical-align: top; text-align: left; }
/* #finaltable * { color: white; background-color: black; } */

/* #finaltable table { border-collapse: collapse; } */
/* #finaltable table, td { border: 1px solid white; } */
#finaltable: hover td { background-color: blue; }
tr:nth-child(odd)    { background-color: #eee; }
tr:nth-child(even)   { background-color: #00FF66; }
input               { text-align: right; border: none; } /* Align input to the right */
textarea           { border: outset; border-color: white; }
table              { border: inset; border-color: white; }

```

```
.hidden      { display: none; }
.unhidden    { display: block; }
table.center { margin-left:auto; margin-right:auto; }
#result      { border-color: red; }
tr.error     { background-color: red; }
pre.output   { background-color: white; }
span.repeated { background-color: red }
span.header  { background-color: blue }
span.comments { background-color: orange }
span.blanks  { background-color: green }
span.nameEqualValue { background-color: cyan }
span.error   { background-color: red }
```

```
body
{
  background-color:#b0c4de; /* blue */
}
```

test/assert.html

```
$ cat test/assert.js
var output = document.getElementById('output');

function assert( outcome, description) {
  var li = document.createElement('li');
  li.className = outcome ? 'pass' : 'fail';
  li.appendChild(document.createTextNode(description));

  output.appendChild(li);
};
```

test/test.css

```
~/jisoncalc(clase)]$ cat test/test.css
.pass:before {
  content: 'PASS: ';
  color: blue;
  font-weight: bold;
}

.fail:before {
  content: 'FAIL: ';
  color: red;
  font-weight: bold;
}
```

test/test.html

```
[~/jisoncalc(clase)]$ cat test/test.html
<!DOCTYPE HTML>
<html lang="en">
  <head>
    <meta charset="UTF-8">
```

```

<title>Testing Our Simple Calculator</title>
<link rel="stylesheet" href="test.css" />
<script type="text/javascript" src="../calculator.js"></script>

</head>
<body>
  <h1>Testing Our Simple Calculator
  </h1>

  <ul id="output"></ul>
  <script type="text/javascript" src="_____.js"></script>

  <script type="text/javascript">
    var r = _____.parse("a = 4*8");
    assert(_____, "a is 4*8");
    assert(_____, "32 == 4*8");
    r = calculator.parse("a = 4;\nb=a+1;\nc=b*2");
    assert(_____, "4 is the first computed result ");
    assert(_____, "a is 4");
    assert(_____, "b is 5");
    assert(_____, "c is 10");
  </script>
  See the NetTuts+ tutorial at <a href="http://net.tutsplus.com/tutorials/javascript-ajax/"
</body>
</html>

```

4.3.1. Práctica: Calculadora con Listas de Expresiones y Variables

Modifique la calculadora vista en la sección anterior 4.3 para que el lenguaje cumpla los siguientes requisitos:

- Extienda el lenguaje de la calculadora para que admita expresiones de asignación $a = 2*3$
- Extienda el lenguaje de la calculadora para que admita listas de sentencias $a = 2; b = a + 1$
- El analizador devuelve la lista de expresiones evaluadas y la tabla de símbolos (con las parejas variable-valor).
- Emita un mensaje de error específico si se intentan modificar las constantes π y e .
- Emita un mensaje de error específico si se intenta una división por cero
- Emita un mensaje de error específico si se intenta acceder para lectura a una variable no inicializada $a = c$
- El lenguaje debería admitir expresiones vacías, estos es secuencias consecutivas de puntos y comas sin producir error ($a = 4;;; b = 5$)
- Introduzca pruebas unitarias como las descritas en la sección ?? (*Quick Tip: Quick and Easy JavaScript Test*)

4.4. Conceptos Básicos del Análisis LR

Los analizadores generados por `jison` entran en la categoría de analizadores *LR*. Estos analizadores construyen una derivación a derechas inversa (o *antiderivación*). De ahí la *R* en *LR* (del inglés *rightmost derivation*). El árbol sintáctico es construido de las hojas hacia la raíz, siendo el último paso en la antiderivación la construcción de la primera derivación desde el símbolo de arranque.

Empezaremos entonces considerando las frases que pueden aparecer en una derivación a derechas. Tales frases constituyen el *lenguaje de las formas sentenciales a derechas FSD*:

Definición 4.4.1. Dada una gramática $G = (\Sigma, V, P, S)$ no ambigua, se denota por *FSD* (*lenguaje de las formas Sentenciales a Derechas*) al lenguaje de las sentencias que aparecen en una derivación a derechas desde el símbolo de arranque.

$$FSD = \left\{ \alpha \in (\Sigma \cup V)^* : \exists S \xRightarrow[RM]{*} \alpha \right\}$$

Donde la notación *RM* indica una derivación a derechas (rightmost). Los elementos de *FSD* se llaman “formas sentenciales derechas”.

Dada una gramática no ambigua $G = (\Sigma, V, P, S)$ y una frase $x \in L(G)$ el proceso de antiderivación consiste en encontrar la última derivación a derechas que dió lugar a x . Esto es, si $x \in L(G)$ es porque existe una derivación a derechas de la forma

$$S \xRightarrow{*} yAz \Rightarrow ywz = x.$$

El problema es averiguar que regla $A \rightarrow w$ se aplicó y en que lugar de la cadena x se aplicó. En general, si queremos antiderivar una forma sentencial derecha $\beta\alpha w$ debemos averiguar por que regla $A \rightarrow \alpha$ seguir y en que lugar de la forma (después de β en el ejemplo) aplicarla.

$$S \xRightarrow{*} \beta Aw \Rightarrow \beta\alpha w.$$

La pareja formada por la regla y la posición se denomina *handle*, *mango* o *manecilla* de la forma. Esta denominación viene de la visualización gráfica de la regla de producción como una mano que nos permite escalar hacia arriba en el árbol. Los “dedos” serían los símbolos en la parte derecha de la regla de producción.

Definición 4.4.2. Dada una gramática $G = (\Sigma, V, P, S)$ no ambigua, y dada una forma sentencial derecha $\alpha = \beta\gamma x$, con $x \in \Sigma^*$, el mango o handle de α es la última producción/posición que dió lugar a α :

$$S \xRightarrow[RM]{*} \beta Bx \Rightarrow \beta\gamma x = \alpha$$

Escribiremos: $handle(\alpha) = (B \rightarrow \gamma, \beta\gamma)$. La función *handle* tiene dos componentes: $handle_1(\alpha) = B \rightarrow \gamma$ y $handle_2(\alpha) = \beta\gamma$

Si dispusiéramos de un procedimiento que fuera capaz de identificar el mango, esto es, de detectar la regla y el lugar en el que se posiciona, tendríamos un mecanismo para construir un analizador. Lo curioso es que, a menudo es posible encontrar un autómata finito que reconoce el lenguaje de los prefijos $\beta\gamma$ que terminan en el mango. Con mas precisión, del lenguaje:

Definición 4.4.3. El conjunto de prefijos viables de una gramática G se define como el conjunto:

$$PV = \left\{ \delta \in (\Sigma \cup V)^* : \exists S \xRightarrow[RM]{*} \alpha = \beta\gamma x \text{ y } \delta \text{ es un prefijo de } handle_2(\alpha) = \beta\gamma \right\}$$

Esto es, el lenguaje de los prefijos viables es el conjunto de frases que son prefijos de $handle_2(\alpha) = \beta\gamma$, siendo α una forma sentencial derecha ($\alpha \in FSD$). Los elementos de *PV* se denominan prefijos viables.

Obsérvese que si se dispone de un autómata que reconoce *PV* entonces se dispone de un mecanismo para investigar el lugar y el aspecto que pueda tener el mango. Si damos como entrada la sentencia $\alpha = \beta\gamma x$ a dicho autómata, el autómata aceptará la cadena $\beta\gamma$ pero rechazará cualquier extensión del prefijo. Ahora sabemos que el mango será alguna regla de producción de G cuya parte derecha sea un sufijo de $\beta\gamma$.

Definición 4.4.4. El siguiente autómata finito no determinista puede ser utilizado para reconocer el lenguaje de los prefijos viables PV:

- Alfabeto = $V \cup \Sigma$
- Los estados del autómata se denominan $LR(0)$ items. Son parejas formadas por una regla de producción de la gramática y una posición en la parte derecha de la regla de producción. Por ejemplo, $(E \rightarrow E + E, 2)$ sería un $LR(0)$ item para la gramática de las expresiones.

Conjunto de Estados:

$$Q = \{(A \rightarrow \alpha, n) : A \rightarrow \alpha \in P, n \leq |\alpha|\}$$

La notación $|\alpha|$ denota la longitud de la cadena α . En vez de la notación $(A \rightarrow \alpha, n)$ escribiremos: $A \rightarrow \beta \uparrow \gamma = \alpha$, donde la flecha ocupa el lugar indicado por el número $n = |\beta|$:

- La función de transición intenta conjeturar que partes derechas de reglas de producción son viables. El conjunto de estados actual del NFA representa el conjunto de pares (regla de producción, posición en la parte derecha) que tienen alguna posibilidad de ser aplicadas de acuerdo con la entrada procesada hasta el momento:

$$\delta(A \rightarrow \alpha \uparrow X \beta, X) = A \rightarrow \alpha X \uparrow \beta \quad \forall X \in V \cup \Sigma$$

$$\delta(A \rightarrow \alpha \uparrow B \beta, \epsilon) = B \rightarrow \uparrow \gamma \quad \forall B \rightarrow \gamma \in P$$

- Estado de arranque: Se añade la “superregla” $S' \rightarrow S$ a la gramática $G = (\Sigma, V, P, S)$. El $LR(0)$ item $S' \rightarrow \uparrow S$ es el estado de arranque.
- Todos los estados definidos (salvo el de muerte) son de aceptación.

Denotaremos por $LR(0)$ a este autómata. Sus estados se denominan $LR(0)$ – items. La idea es que este autómata nos ayuda a reconocer los prefijos viables PV.

Una vez que se tiene un autómata que reconoce los prefijos viables es posible construir un analizador sintáctico que construye una antiderivación a derechas. La estrategia consiste en “alimentar” el autómata con la forma sentencial derecha. El lugar en el que el autómata se detiene, rechazando indica el lugar exacto en el que termina el *handle* de dicha forma.

Ejemplo 4.4.1. Consideremos la gramática:

$$\begin{aligned} S &\rightarrow a S b \\ S &\rightarrow \epsilon \end{aligned}$$

El lenguaje generado por esta gramática es $L(G) = \{a^n b^n : n \geq 0\}$ Es bien sabido que el lenguaje $L(G)$ no es regular. La figura 4.1 muestra el autómata finito no determinista con ϵ -transiciones (NFA) que reconoce los prefijos viables de esta gramática, construido de acuerdo con el algoritmo 4.4.4.

Véase <https://github.com/crguezl/jison-aSb> para una implementación en Jison de una variante de esta gramática.

Ejercicio 4.4.1. Simule el comportamiento del autómata sobre la entrada *aabb*. ¿Donde rechaza? ¿En que estados está el autómata en el momento del rechazo?. ¿Qué etiquetas tienen? Haga también las trazas del autómata para las entradas *aaSbb* y *aSb*. ¿Que antiderivación ha construido el autómata con sus sucesivos rechazos? ¿Que terminales se puede esperar que hayan en la entrada cuando se produce el rechazo del autómata?



Figura 4.1: NFA que reconoce los prefijos viables

4.5. Construcción de las Tablas para el Análisis SLR

4.5.1. Los conjuntos de Primeros y Siguietes

Repasemos las nociones de conjuntos de *Primeros* y *siguietes*:

Definición 4.5.1. Dada una gramática $G = (\Sigma, V, P, S)$ y una frase $\alpha \in (V \cup \Sigma)^*$ se define el conjunto $FIRST(\alpha)$ como:

$$FIRST(\alpha) = \{b \in \Sigma : \alpha \xRightarrow{*} b\beta\} \cup N(\alpha)$$

donde:

$$N(\alpha) = \begin{cases} \{\epsilon\} & \text{si } \alpha \xRightarrow{*} \epsilon \\ \emptyset & \text{en otro caso} \end{cases}$$

Definición 4.5.2. Dada una gramática $G = (\Sigma, V, P, S)$ y una variable $A \in V$ se define el conjunto $FOLLOW(A)$ como:

$$FOLLOW(A) = \{b \in \Sigma : \exists S \xRightarrow{*} \alpha A b\beta\} \cup E(A)$$

donde

$$E(A) = \begin{cases} \{\$ \} & \text{si } S \xRightarrow{*} \alpha A \\ \emptyset & \text{en otro caso} \end{cases}$$

Algoritmo 4.5.1. Construcción de los conjuntos $FIRST(X)$

1. Si $X \in \Sigma$ entonces $FIRST(X) = X$
2. Si $X \rightarrow \epsilon$ entonces $FIRST(X) = FIRST(X) \cup \{\epsilon\}$
3. Si $X \in V$ y $X \rightarrow Y_1 Y_2 \cdots Y_k \in P$ entonces

$i = 1;$

do

$FIRST(X) = FIRST(X) \cup FIRST(Y_i) - \{\epsilon\};$

$i++;$

mientras ($\epsilon \in FIRST(Y_i)$ and $(i \leq k)$)

si ($\epsilon \in FIRST(Y_k)$ and $i > k$) $FIRST(X) = FIRST(X) \cup \{\epsilon\}$

Este algoritmo puede ser extendido para calcular $FIRST(\alpha)$ para $\alpha = X_1 X_2 \cdots X_n \in (V \cup \Sigma)^*$.

Algoritmo 4.5.2. *Construcción del conjunto $FIRST(\alpha)$*

```

 $i = 1;$ 
 $FIRST(\alpha) = \emptyset;$ 
do
     $FIRST(\alpha) = FIRST(\alpha) \cup FIRST(X_i) - \{\epsilon\};$ 
     $i ++;$ 
mientras ( $\epsilon \in FIRST(X_i)$  and ( $i \leq n$ ))
si ( $\epsilon \in FIRST(X_n)$  and  $i > n$ )  $FIRST(\alpha) = FIRST(X) \cup \{\epsilon\}$ 

```

Algoritmo 4.5.3. *Construcción de los conjuntos $FOLLOW(A)$ para las variables sintácticas $A \in V$: Repetir los siguientes pasos hasta que ninguno de los conjuntos $FOLLOW$ cambie:*

1. $FOLLOW(S) = \{\$$ ($\$$ representa el final de la entrada)
2. Si $A \rightarrow \alpha B \beta$ entonces

$$FOLLOW(B) = FOLLOW(B) \cup (FIRST(\beta) - \{\epsilon\})$$

3. Si $A \rightarrow \alpha B$ o bien $A \rightarrow \alpha B \beta$ y $\epsilon \in FIRST(\beta)$ entonces

$$FOLLOW(B) = FOLLOW(B) \cup FOLLOW(A)$$

4.5.2. Construcción de las Tablas

Para la construcción de las tablas de un analizador SLR se construye el *autómata finito determinista* (DFA) (Q, Σ, δ, q_0) equivalente al NFA presentado en la sección 4.4 usando el *algoritmo de construcción del subconjunto*.

Como recordará, en la construcción del subconjunto, partiendo del estado de arranque q_0 del NFA con ϵ -transiciones se calcula su *clausura* $\overline{\{q_0\}}$ y las clausuras de los conjuntos de estados $\delta(\overline{\{q_0\}}, a)$ a los que transita. Se repite el proceso con los conjuntos resultantes hasta que no se introducen nuevos conjuntos-estado.

La clausura \overline{A} de un subconjunto de estados del autómata A esta formada por todos los estados que pueden ser alcanzados mediante transiciones etiquetadas con la palabra vacía (denominadas ϵ transiciones) desde los estados de A . Se incluyen en \overline{A} , naturalmente los estados de A .

$$\overline{A} = \{q \in Q / \exists q' \in A : \hat{\delta}(q', \epsilon) = q\}$$

Aquí $\hat{\delta}$ denota la *función de transición del autómata* extendida a cadenas de Σ^* .

$$\hat{\delta}(q, x) = \begin{cases} \delta(\hat{\delta}(q, y), a) & \text{si } x = ya \\ q & \text{si } x = \epsilon \end{cases} \quad (4.1)$$

En la práctica, y a partir de ahora así lo haremos, se prescinde de diferenciar entre δ y $\hat{\delta}$ usándose indistintamente la notación δ para ambas funciones.

La clausura puede ser computada usando una estructura de pila o aplicando la expresión recursiva dada en la ecuación 4.1.

Para el NFA mostrado en el ejemplo 4.4.1 el DFA construido mediante esta técnica es el que se muestra en la figura 4.3. Se ha utilizado el símbolo # como marcador. Se ha omitido el número 3 para que los estados coincidan en numeración con los generados por `jison` (véase el cuadro ??).



Figura 4.2: DFA equivalente al NFA de la figura 4.1

Un analizador sintáctico LR utiliza una tabla para su análisis. Esa tabla se construye a partir de la tabla de transiciones del DFA. De hecho, la tabla se divide en dos tablas, una llamada *tabla de saltos* o *tabla de gotos* y la otra *tabla de acciones*.

La tabla *goto* de un analizador *SLR* no es más que la tabla de transiciones del autómata DFA obtenido aplicando la construcción del subconjunto al NFA definido en 4.4.4. De hecho es la tabla de transiciones restringida a V (recuerde que el alfabeto del autómata es $V \cup \Sigma$, i denota al i -ésimo estado resultante de aplicar la construcción del subconjunto y que I_i denota al conjunto de LR(0) item asociado con dicho estado):

$$\delta_{|V \times Q} : V \times Q \rightarrow Q.$$

donde se define $goto(i, A) = \delta(A, I_i)$

La parte de la función de transiciones del DFA que corresponde a los terminales que no producen rechazo, esto es, $\delta_{|\Sigma \times Q} : \Sigma \times Q \rightarrow Q$ se adjunta a una tabla que se denomina *tabla de acciones*. La tabla de acciones es una tabla de doble entrada en los estados y en los símbolos de Σ . Las acciones de transición ante terminales se denominan *acciones de desplazamiento* o (*acciones shift*):

$$\delta_{|\Sigma \times Q} : \Sigma \times Q \rightarrow Q$$

donde se define $action(i, a) = shift \delta(a, I_i)$

Cuando un estado s contiene un LR(0)-item de la forma $A \rightarrow \alpha \uparrow$, esto es, el estado corresponde a un posible rechazo, ello indica que hemos llegado a un final del prefijo viable, que hemos visto α y que, por tanto, es probable que $A \rightarrow \alpha$ sea el *handle* de la forma sentencial derecha actual. Por tanto, añadiremos en entradas de la forma (s, a) de la tabla de acciones una acción que indique que hemos encontrado el mango en la posición actual y que la regla asociada es $A \rightarrow \alpha$. A una acción de este tipo se la denomina *acción de reducción*.

La cuestión es, ¿para que valores de $a \in \Sigma$ debemos disponer que la acción para (s, a) es de reducción?

Se define $action(i, a) = reduce A \rightarrow \alpha$ ¿Pero, para que $a \in \Sigma$?

Podríamos decidir que ante cualquier terminal $a \in \Sigma$ que produzca un rechazo del autómata, pero podemos ser un poco mas selectivos. No cualquier terminal puede estar en la entrada en el momento en el que se produce la antiderivación o reducción. Observemos que si $A \rightarrow \alpha$ es el *handle* de γ es porque:

$$\begin{array}{ccc} & * & * \\ \exists S & \xRightarrow{RM} & \beta A b x \xRightarrow{RM} \beta \alpha b x = \gamma \end{array}$$

Por tanto, cuando estamos reduciendo por $A \rightarrow \alpha$ los únicos terminales legales que cabe esperar en una reducción por $A \rightarrow \alpha$ son los terminales $b \in FOLLOW(A)$.

Se define $action(i, b) = reduce\ A \rightarrow \alpha$ Para $b \in FOLLOW(A)$

Dada una gramática $G = (\Sigma, V, P, S)$, podemos construir las tablas de acciones (*action table*) y transiciones (*gotos table*) mediante el siguiente algoritmo:

Algoritmo 4.5.4. *Construcción de Tablas SLR*

1. Utilizando el Algoritmo de Construcción del Subconjunto, se construye el Autómata Finito Determinista (DFA) $(Q, V \cup \Sigma, \delta, I_0, F)$ equivalente al Autómata Finito No Determinista (NFA) definido en 4.4.4. Sea $C = \{I_1, I_2, \dots, I_n\}$ el conjunto de estados del DFA. Cada estado I_i es un conjunto de $LR(0)$ -items o estados del NFA. Asociemos un índice i con cada conjunto I_i .
2. La tabla de gotos no es más que la función de transición del autómata restringida a las variables de la gramática:

$$goto(i, A) = \delta(I_i, A) \text{ para todo } A \in V$$

3. Las acciones para el estado I_i se determinan como sigue:

a) Si $A \rightarrow \alpha \uparrow a \beta \in I_i$, $\delta(I_i, a) = I_j$, $a \in \Sigma$ entonces:

$$action[i][a] = shift\ j$$

b) Si $S' \rightarrow S \uparrow \in I_i$ entonces

$$action[i][\$] = accept$$

c) Para cualquier otro caso de la forma $A \rightarrow \alpha \uparrow \in I_i$ distinto del anterior hacer

$$\forall a \in FOLLOW(A) : action[i][a] = reduce\ A \rightarrow \alpha$$

4. Las entradas de la tabla de acción que queden indefinidas después de aplicado el proceso anterior corresponden a acciones de “error”.

Definición 4.5.3. Si alguna de las entradas de la tabla resulta multievaluada, decimos que existe un conflicto y que la gramática no es SLR.

1. En tal caso, si una de las acciones es de “reducción” y la otra es de “desplazamiento”, decimos que hay un conflicto shift-reduce o conflicto de desplazamiento-reducción.
2. Si las dos reglas indican una acción de reducción, decimos que tenemos un conflicto reduce-reduce o de reducción-reducción.

Ejemplo 4.5.1. Al aplicar el algoritmo 4.5.4 a la gramática 4.4.1

1	$S \rightarrow a S b$
2	$S \rightarrow \epsilon$

partiendo del autómata finito determinista que se construyó en la figura 4.3 y calculando los conjuntos de primeros y siguientes

	<i>FIRST</i>	<i>FOLLOW</i>
<i>S</i>	<i>a</i> , ϵ	<i>b</i> , $\$$

obtenemos la siguiente tabla de acciones SLR:

	<i>a</i>	<i>b</i>	$\$$
0	<i>s2</i>	<i>r2</i>	<i>r2</i>
1			aceptar
2	<i>s2</i>	<i>r2</i>	<i>r2</i>
4		<i>s5</i>	
5		<i>r1</i>	<i>r1</i>

Las entradas denotadas con *s n* (*s* por shift) indican un desplazamiento al estado *n*, las denotadas con *r n* (*r* por reduce o reducción) indican una operación de reducción o antiderivación por la regla *n*. Las entradas vacías corresponden a acciones de error.

El método de análisis *LALR* usado por `jison` es una extensión del método SLR esbozado aquí. Supone un compromiso entre potencia (conjunto de gramáticas englobadas) y eficiencia (cantidad de memoria utilizada, tiempo de proceso). Veamos como `jison` aplica la construcción del subconjunto a la gramática del ejemplo 4.4.1. Para ello construimos el siguiente programa `jison`:

```
[~/srcPLgrado/aSb(develop)]$ cat -n aSb.jison
 1 %lex
 2 %%
 3 .          { return yytext; }
 4 /lex
 5 %%
 6 P: S        { return $1; }
 7 ;
 8 S: /* empty */ { console.log("empty"); $$ = ''; }
 9   | 'a' S 'b' { console.log("S -> aSb"); $$ = $1+$2+$3; }
10 ;
11 %%
```

y lo compilamos con `jison`. Estas son las opciones disponibles:

```
nereida:[~/PLgradoBOOK(eps)]$ jison --help
```

```
Usage: jison [file] [lexfile] [options]
```

```
file          file containing a grammar
lexfile       file containing a lexical grammar
```

Options:

```
-o FILE, --outfile FILE      Filename and base module name of the generated parser
-t, --debug                  Debug mode
-t TYPE, --module-type TYPE  The type of module to generate (commonjs, amd, js)
-V, --version                 print version and exit
```

Desafortunadamente carece de la típica opción `-v` que permite generar las tablas de análisis. Podemos intentar usar `bison`, pero, obviamente, `bison` protesta ante la entrada:

```
[~/srcPLgrado/aSb(develop)]$ bison -v aSb.jison
aSb.jison:1.1-4: invalid directive: '%lex'
aSb.jison:3.1: syntax error, unexpected identifier
aSb.jison:4.1: invalid character: '/'
```

El error es causado por la presencia del analizador léxico empotrado en el fichero aSb.jison. Si suprimimos provisionalmente las líneas del analizador léxico empotrado, bison es capaz de analizar la gramática:

```
[~/srcPLgrado/aSb(develop)]$ bison -v aSb.jison
[~/srcPLgrado/aSb(develop)]$ ls -ltr | tail -1
-rw-rw-r-- 1 casiano staff 926 19 mar 13:29 aSb.output
```

Que tiene los siguientes contenidos:

```
[~/srcPLgrado/aSb(develop)]$ cat -n aSb.output
 1 Grammar
 2
 3     0 $accept: P $end
 4
 5     1 P: S
 6
 7     2 S: /* empty */
 8     3 | 'a' S 'b'
 9
10
11 Terminals, with rules where they appear
12
13 $end (0) 0
14 'a' (97) 3
15 'b' (98) 3
16 error (256)
17
18
19 Nonterminals, with rules where they appear
20
21 $accept (5)
22     on left: 0
23 P (6)
24     on left: 1, on right: 0
25 S (7)
26     on left: 2 3, on right: 1 3
27
28
29 state 0
30
31     0 $accept: . P $end
32
33     'a' shift, and go to state 1
34
35     $default reduce using rule 2 (S)
36
37     P go to state 2
38     S go to state 3
39
```



```

40
41 state 1
42
43     3 S: 'a' . S 'b'
44
45     'a' shift, and go to state 1
46
47     $default reduce using rule 2 (S)
48
49     S go to state 4
50
51
52 state 2
53
54     0 $accept: P . $end
55
56     $end shift, and go to state 5
57
58
59 state 3
60
61     1 P: S .
62
63     $default reduce using rule 1 (P)
64
65
66 state 4
67
68     3 S: 'a' S . 'b'
69
70     'b' shift, and go to state 6
71
72
73 state 5
74
75     0 $accept: P $end .
76
77     $default accept
78
79
80 state 6
81
82     3 S: 'a' S 'b' .
83
84     $default reduce using rule 3 (S)

```

Observe que el final de la entrada se denota por `$end` y el marcador en un LR-item por un punto. Fíjese en el estado 1: En ese estado están también los items

$$S \rightarrow . 'a' S 'b' \text{ y } S \rightarrow .$$

sin embargo no se explicitan por que se entiende que su pertenencia es consecuencia directa de aplicar la operación de clausura. Los LR items cuyo marcador no está al principio se denominan *items núcleo*.

4.6. Práctica: Traducción de Infijo a Postfijo

Modifique el programa Jison realizado en la práctica 4.3.1 para traducir de infijo a postfijo. Añada los operadores de comparación e igualdad. Por ejemplo

Infijo	Postfijo
a = 3+2*4	3 2 4 * + &a =
b = a == 11	a 11 == &b =

En estas traducciones la notación &a indica la dirección de la variable a y a indica el valor almacenado en la variable a.

Añada sentencias `if ... then` e `if ... then ... else`

Para realizar la traducción de estas sentencias añada instrucciones `jmp label` y `jmpz label` (por *jump if zero*) y etiquetas:

Infijo	Postfijo
a = (2+5)*3;	2 5 + 3 * &a = a 0 == jmpz else1
if a == 0 then b = 5 else b = 3;	5 &b = jmp endif0
c = b + 1;	:else1 3 &b = :endif0 b 1 + &c =

Introduzca pruebas unitarias como las descritas en la sección ?? (*Quick Tip: Quick and Easy JavaScript Testing w*

```
[~/srcPLgrado/jisoninfix2postfix(master)]$ cat test/test.html
<!DOCTYPE HTML>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <title>Testing Our Simple Translator</title>
    <link rel="stylesheet" href="test.css" />
    <script type="text/javascript" src="../calculator.js"></script>
```

```

</head>
<body>
  <h1>Testing Our Simple Translator
  </h1>

  <ul id="output"></ul>
  <script type="text/javascript" src="assert.js"></script>

  <script type="text/javascript">

    var r = calculator.parse("a = 4*8");
    assert( /4\s*8\s*[*]\s*a\s*=\s*/.exec(r), "a is 4*8");

    r = calculator.parse("a=4;b=a+1");
    r = r.replace(/\s+/g, '');
    var expected = "4a=a1+b=";
    assert( r == expected, "a = 4;\nb=a+1 translated");

    var r = calculator.parse("if a > 0 then b = 1 else b = 2");
    r = r.replace(/\s+/g, '');
    expected = "a 0 > jmpz else1 1 b = jmp endif0 :else1 2 b = :endif0".
      replace(/\s+/g, '');
    assert( r == expected, "'if a > 0 then b = 1 else b = 2' translated");
  </script>
  See the NetTuts+ tutorial at <a href="http://net.tutsplus.com/tutorials/javascript-ajax/"
</body>
</html>

```

4.7. Práctica: Calculadora con Funciones

Añada funciones y sentencias de llamada a función a la práctica de traducción de infijo a postfijo

4.6. Sigue un ejemplo de traducción:

```
def f(x) { x + 1 }
def g(a, b) { a * f(b) }
c = 3;
f(1+c);
g(3, 4)
```

```
f      args :x
      $x
      1
      +
      return
:g     args :a,:b
      $a
      $b
      call :f
      *
      return
:main:
      3
      &c
      =
      1
      c
      +
      call :f
      3
      4
      call :g
```

- Las funciones retornan la última expresión evaluada
- Es un error llamar a una función con un número de argumentos distinto que el número de parámetros con el que fue declarada
- En la llamada, los argumentos se empujan en la pila. Después la instrucción `call :etiqueta` llama a la función con el nombre dado por `laetiqueta`
- Dentro de la función los argumentos se sitúan por encima del puntero base. La pseudo-instrucción `args`, `p1`, `p2`, ... da nombre a los parámetros empujados. Dentro del cuerpo de la función nos referimos a ellos prefijándolos con `$`.
- La instrucción `return` limpia la pila dejándola en su estado anterior y retorna la última expresión evaluada

4.8. Práctica: Calculadora con Análisis de Ámbito

Extienda la práctica anterior para que haga un análisis completo del ámbito de las variables.

- Añada declaraciones de variable con `var x`, `y = 1`, `z`. Las variables podrán opcionalmente ser inicializadas. Se considerará un error usar una variable no declarada.
- Modifique la gramática para que permita el anidamiento de funciones: funciones dentro de funciones.

```
var c = 4, d = 1, e;
def g(a, b) {
  var d, e;
  def f(u, v) { a + u + v + d }
  a * f(b, 2) + d + c
}
```

- Una declaración de variable en un ámbito anidado tapa a una declaración con el mismo nombre en el ámbito exterior.

<pre> var c = 4, d = 1, e; def g(a, b) { var d, e; # esta "d" tapa la d anterior def f(u, v) { a + u + v + d } a * f(b, 2) + d + c } </pre>	<pre> # global: var c,d,e :g.f \$a, 1 \$u, 0 + \$v, 0 + d, 1 + return :g \$a, 0 \$b, 0 2 call :g.f * d, 0 # acceder a la d en el ámbito actual + c, 1 + return </pre>
---	---

- Los nombres de funciones se traducen por una secuencia anidada de nombres que indican su ámbito. Así la función *f* anidada en *g* es traducida a la función con nombre *g.f*. Una función *h* anidada en una función *f* anidada en *g* es traducida a la función con nombre *g.f.h*
- Las variables además de su nombre (dirección/offset) reciben un entero adicional 0,1,2, ... que indica su nivel de anidamiento. El número de stack frames que hay que recorrer para llegar a la variable

```

$a, 1
$u, 0
+
$v, 0
+
d, 1
+

```

Así *\$a, 1* significa acceder al parámetro *a* que está a distancia 1 del stack frame/ámbito actual y *\$v, 0* es el parámetro *v* en el ámbito/stack frame actual

- El frame pointer o base pointer BP indica el nivel de anidamiento esático (en el fuente) de la rutina. Así cuando se va a buscar una variable local declarada en la rutina que anida la actual se recorre la lista de frames via BP o frame pointer tantas veces como el nivel de anidamiento indique.
- 1. Esto es lo que dice la Wikipedia sobre la implementación de llamadas a subrutinas anidadas:

Programming languages that support nested subroutines also have a field in the call frame that points to the stack frame of the latest activation of the procedure that most closely encapsulates the callee, i.e. the immediate scope of the callee. This is

called an access link or static link (as it keeps track of static nesting during dynamic and recursive calls) and provides the routine (as well as any other routines it may invoke) access to the local data of its encapsulating routines at every nesting level.

2. Esto es lo que dice sobre las ventajas de tener una pila y de almacenar la dirección de retorno y las variables locales:

When a subroutine is called, the location (address) of the instruction at which it can later resume needs to be saved somewhere. Using a stack to save the return address has important advantages over alternatives. One is that each task has its own stack, and thus the subroutine can be *reentrant*, that is, can be active simultaneously for different tasks doing different things. Another benefit is that *recursion* is automatically supported. When a function calls itself recursively, a return address needs to be stored for each activation of the function so that it can later be used to return from the function activation. This capability is automatic with a stack.

3. Almacenamiento local:

A subroutine frequently needs memory space for storing the values of local variables, the variables that are known only within the active subroutine and do not retain values after it returns. It is often convenient to allocate space for this use by simply moving the top of the stack by enough to provide the space. This is very fast compared to heap allocation. Note that each separate activation of a subroutine gets its own separate space in the stack for locals.

4. Parámetros:

Subroutines often require that values for parameters be supplied to them by the code which calls them, and it is not uncommon that space for these parameters may be laid out in the call stack.

The call stack works well as a place for these parameters, especially since each call to a subroutine, which will have differing values for parameters, will be given separate space on the call stack for those values.

5. Pila de Evaluación

Operands for arithmetic or logical operations are most often placed into registers and operated on there. However, in some situations the operands may be stacked up to an arbitrary depth, which means something more than registers must be used (this is the case of *register spilling*). The stack of such operands, rather like that in an RPN calculator, is called an *evaluation stack*, and may occupy space in the call stack.

6. Puntero a la instancia actual

Some object-oriented languages (e.g., C++), store the **this** pointer along with function arguments in the call stack when invoking methods. The **this pointer** points to the object instance associated with the method to be invoked.

- Los parámetros se siguen prefijando de \$ como en la práctica anterior

- Sigue un ejemplo de traducción:

<pre> var c = 4, d = 1, e; def f(x) { var y = 1; x + y } def g(a, b) { var d, e; def f(u, v) { a + u + v + d } a * f(b, 2) + d + c } c = 3; f(1+c); g(3, 4) </pre>	<pre> # global: var c, # f: args x # f: var y :f 1 &y, 0 = \$x, 0 y, 0 + return # g: args a,b # g: var d,e # g.f: args u,v :g.f \$a, 1 \$u, 0 + \$v, 0 + d, 1 + return :g \$a, 0 \$b, 0 2 call :g.f * d, 0 + c, 1 + return :main: 4 &c, 0 = 1 &d, 0 = 3 &c, 0 = 1 c, 0 + call :f 3 4 call :g </pre>
--	--

- Puede comenzar haciendo un fork del proyecto `ull-etsii-grado-pl-infix2postfix` en GitHub. Esta incompleto. Rellene las acciones semánticas que faltan; la mayoría relacionadas con el análisis de ámbito
- Veanse:
 - Véase COMP 3290 Compiler Construction Fall 2008 Notes/Symbol Tables
 - El capítulo Symbol Table Structure del libro de Muchnick Advanced Compiler Design Implementation [5]
 - El capítulo Symbol Table Structure del libro de Basics of Compiler Design de Torben Ægidius Mogensen [6]

4.9. Algoritmo de Análisis LR

Así pues la tabla de transiciones del autómata nos genera dos tablas: la tabla de acciones y la de saltos. El algoritmo de análisis sintáctico *LR* en el que se basa *jison* utiliza una pila y dos tablas para analizar la entrada. Como se ha visto, la tabla de acciones contiene cuatro tipos de acciones:

1. Desplazar (*shift*)
2. Reducir (*reduce*)
3. Aceptar
4. Error

El algoritmo utiliza una pila en la que se guardan los estados del autómata. De este modo se evita tener que “comenzar” el procesamiento de la forma sentencial derecha resultante después de una reducción (antiderivación).

Algoritmo 4.9.1. *Análizador LR*

```

push(s0);
b = yylex();
for( ; ; ) {
  s = top(0); a = b;
  switch (action[s][a]) {
    case "shift t" :
      t.attr = a.attr;
      push(t);
      b = yylex();
      break;
    case "reduce A ->alpha" :
      eval(Sem{A -> alpha}(top(|alpha|-1).attr, ... , top(0).attr));
      pop(|alpha|);
      push(goto[top(0)][A]);
      break;
    case "accept" : return (1);
    default : yyerror("syntax error");
  }
}

```

- Como es habitual, $|x|$ denota la longitud de la cadena x .
- La función `top(k)` devuelve el elemento que ocupa la posición k desde el *top* de la pila (esto es, está a profundidad k).

- La función `pop(k)` extrae `k` elementos de la pila.
- La notación `state.attr` hace referencia al atributo asociado con cada estado, el cual desde el punto de vista del programador esta asociado con el correspondiente símbolo de la parte derecha de la regla. Nótese que cada estado que está en la pila es el resultado de una transición con un símbolo. El atributo de ese símbolo es guardado en el objeto estado cada vez que ocurre una transición.
- Denotamos por `Sem {reduce A -> alpha}` el código de la acción semántica asociada con la regla $A \rightarrow \alpha$.

Todos los analizadores LR comparten, salvo pequeñas excepciones, el mismo algoritmo de análisis. Lo que más los diferencia es la forma en la que construyen las tablas. En `jison` la construcción de las tablas de *acciones* y *gotos* se realiza por defecto mediante el algoritmo *LALR*.

4.10. El módulo Generado por jison

4.10.1. Version

En esta sección estudiamos el analizador generado por Jison:

```
[~/Dropbox/src/javascript/PLgrado/jison-aSb(develop)]$ jison --version
0.4.2
```

4.10.2. Gramática Inicial

Veamos el módulo generado por jison para esta gramática:

```
[~/srcPLgrado/aSb(develop)]$ cat aSb.jison
%lex
%%
.          { return yytext; }
/lex
%%
S: /* empty */ { console.log("empty"); }
  | 'a' S 'b' { console.log("S -> aSb"); }
;
%%
```

4.10.3. Tablas

Esta es la primera parte del parser generado:

```
/* parser generated by jison 0.4.2 */
var aSb = (function() {
  var parser = {
    trace: function trace() {},
    yy: {},
    symbols_: {
      "$accept": 0, /* super-arranque $accept -> S */
      "$end": 1    /* end of input */
      "error": 2, /* numero para el símbolo 'error' */
      "S": 3,     /* numero para el símbolo 'S' */
      "a": 4,
      "b": 5,
    },
  },
```

```

/* array inverso de terminales */
terminals_: { /* numero -> terminal */
    2: "error",
    4: "a",
    5: "b"
},
productions_:
[0,
/* 1 */    [3, 0], /* S : vacio          simbolo,longitud de la parte derecha */
/* 2 */    [3, 3] /* S : a S b          simbolo,longitud */
],

```



Figura 4.3: DFA construido por Jison

4.10.4. Acciones Semánticas

Cada vez que se produce una acción de reducción esta función es llamada:

```
performAction: function anonymous(yytext, yyleng, yylineno, yy, yystate, $$, _$) {  
  
    var $0 = $$.$length - 1;  
    switch (yystate) { /* yystate: numero de regla de producción */  
        case 1:  
            console.log("empty");  
            break;  
        case 2:  
            console.log("S -> aSb");  
            break;  
    }  
},
```

- Parece que cuando se llama a este método **this** refiere a un objeto **yyval**. Este es el punto de llamada a la acción semántica dentro del parser generado por Jison. Puede encontrarse dentro del parser en el caso de un **switch** que corresponde a la acción de reducción:

```
r = this.performAction.call(yyval, yytext, yyleng, yylineno, this.yy, action[1], vstack, .
```

El método **call** nos permite invocar una función como si fuera un método de algún otro objeto. Véase la sección ??.

Este objeto **yyval** tiene dos atributos: **\$** y **_\$**.

- El atributo **\$** se corresponde con **\$\$** de la gramática (atributo de la variable sintactica en la parte izquierda)
 - El atributo **_\$** guarda información sobre la posición del último token leído.
- **yytext** parece contener el texto asociado con el token actual
 - **yyleng** es la longitud del token actual
 - **yylineno** es la línea actual (empezando en 0)
 - **yy** es un objeto con dos atributos **lexer** y **parser**
 - **yystate** es el estado actual
 - **\$\$** parece ser un array/pila conteniendo los valores de los atributos asociados con los estados de la pila (**vstack** ¿Por value stack?)
 - Así pues **\$0** es el índice en **\$0** del último elemento de **\$\$**. Por ejemplo, una acción semántica asociada con una regla **A : B C D** con tres elementos como:

```
$$ = $1 + $2 + $3;
```

Se traduce por:

```
this.$ = $$[$0 - 2] + $$[$0 - 1] + $$[$0];
```

- **_\$** Es un array con la información sobre la localización de los simbolos (**lstack** ¿Por location stack?)

4.10.5. Tabla de Acciones y GOTOS

```
table: [{
/* 0 */    1: [2, 1],    /* En estado 0 viendo $end(1) reducir por S : vacio */
           3: 1,        /* En el estado 0 viendo S(3) ir al estado 1 */
           4: [1, 2]    /* Estado 0 viendo a(4) shift(1) al estado 2 */
}, {
/* 1 */    1: [3]        /* En 1 viendo $end(1) aceptar */
}, {
/* 2 */    3: 3,        /* En 2 viendo S ir a 3 */
           4: [1, 2],    /* En 2 viendo a(4) shift a 2 */
           5: [2, 1]    /* En 2 viendo b(5) reducir por regla 1: S -> vacio */
}, {
/* 3 */    5: [1, 4]    /* En 3 viendo b(5) shift a 4 */
}, {
/* 4 */    1: [2, 2],    /* En 4 viendo $end(1) reducir(2) por la 2: S -> aSb */
           5: [2, 2]    /* En 4 viendo b(5) reducir por la 2: S-> aSb */
}],
```

- La tabla es un array de objetos
- El índice de la tabla es el estado. En el ejemplo tenemos 5 estados
- El objeto/hash que es el valor contiene las acciones ante los símbolos.
 1. Los atributos/claves son los símbolos, los valores las acciones
 2. Las acciones son de dos tipos:
 - a) El número del estado al que se transita mediante la tabla goto cuando el símbolo es una variable sintactica
 - b) Un par [tipo de acción, estado o regla]. Si el tipo de acción es 1 indica un shift al estado con ese número. Si el tipo de acción es 2 indica una reducción por la regla con ese número.
 3. Por ejemplo table[0] es

```
{
    1: [2, 1],    /* En estado 0 viendo $end(1) reducir(2) por S : vacio */
    3: 1,        /* En el estado 0 viendo S(3) ir (goto) al estado 1 */
    4: [1, 2]    /* Estado 0 viendo a(4) shift(1) al estado 2 */
}
```

4.10.6. defaultActions

```
defaultActions: {},
```

- defaultActions contiene las acciones por defecto.
- Después de la construcción de la tabla, Jison identifica para cada estado la reducción que tiene el conjunto de lookaheads mas grande. Para reducir el tamaño del parser, Jison puede decidir suprimir dicho conjunto y asignar esa reducción como acción del parser por defecto. Tal reducción se conoce como *reducción por defecto*.
- Esto puede verse en este segmento del código del parser:

```
while (true) {
    state = stack[stack.length - 1];
    if (this.defaultActions[state]) {
```

```

        action = this.defaultActions[state];
    } else {
        if (symbol === null || typeof symbol == "undefined") {
            symbol = lex();
        }
        action = table[state] && table[state][symbol];
    }
    ...
}

```

4.10.7. Reducciones

```

parse: function parse(input) {
    ...
    while (true) {
        state = stack[stack.length - 1];
        if (this.defaultActions[state]) {
            action = this.defaultActions[state];
        } else {
            if (symbol === null || typeof symbol == "undefined") {
                symbol = lex(); /* obtener siguiente token */
            }
            action = table[state] && table[state][symbol];
        }
        if (typeof action === "undefined" || !action.length || !action[0]) {
            ... // error
        }
        if (action[0] instanceof Array && action.length > 1) {
            throw new Error("Parse Error: multiple actions possible at state: ...")
        }
        switch (action[0]) {
            case 1:                                     // shift
                ...
                break;
            case 2:                                     // reduce
                len = this productions_[action[1]][1]; // longitud de la producción
                yyval.$ = vstack[vstack.length - len];
                yyval._$ = {                             // datos de la posición
                    first_line: lstack[lstack.length - (len || 1)].first_line,
                    last_line: lstack[lstack.length - 1].last_line,
                    first_column: lstack[lstack.length - (len || 1)].first_column,
                    last_column: lstack[lstack.length - 1].last_column
                };
                ...
                r = this.performAction.call(yyval, yytext, yyleng, yylineno, this.yy, action[1]);
                if (typeof r !== "undefined") {
                    return r; /* un return de algo distinto de undefined nos saca del parser */
                }
                if (len) {                               /* retirar de las pilas */
                    stack = stack.slice(0, - 1 * len * 2); /* simbolo, estado, simbolo, estado */
                    vstack = vstack.slice(0, - 1 * len);   /* retirar atributos */
                    lstack = lstack.slice(0, - 1 * len);    /* retirar localizaciones */
                }
                stack.push(this productions_[action[1]][0]); /* empujemos el símbolo */
            }
        }
    }
}

```

```

        vstack.push(yyval.$);                /* empujamos valor semantico */
        lstack.push(yyval._$);                /* empujamos localización */
        newState = table[stack[stack.length - 2]][stack[stack.length - 1]];
        stack.push(newState);                /* empujamos goto[top][A]*/
        break;
    case 3: // accept
        return true;
    }
}
return true;
}

```

4.10.8. Desplazamientos/Shifts

```

parse: function parse(input) {
    ...
    while (true) {
        state = stack[stack.length - 1];    /* estado en el top de la pila */
        if (this.defaultActions[state]) {    /* definida la acción por defecto? */
            action = this.defaultActions[state];
        } else {
            if (symbol === null || typeof symbol == "undefined") {
                symbol = lex();                /* obtener token */
            }
            action = table[state] && table[state][symbol]; /* obtener la acción para el estado */
        }
        if (typeof action === "undefined" || !action.length || !action[0]) {
            ... /* error */
        }
        if (action[0] instanceof Array && action.length > 1) {
            throw new Error("Parse Error: multiple actions possible at state: " + state + ", t
        }
        switch (action[0]) {
            case 1:
                stack.push(symbol);                /* empujamos token */
                vstack.push(this.lexer.yytext);    /* empujamos el atributo del token */
                lstack.push(this.lexer.yylloc);    /* salvamos la localización del token */
                stack.push(action[1]);              /* salvamos el estado */
                symbol = null;
                if (!preErrorSymbol) {              /* si no hay errores ... */
                    yyleng = this.lexer.yyleng;    /* actualizamos los atributos */
                    yytext = this.lexer.yytext;    /* del objeto */
                    yylineno = this.lexer.yylineno;
                    yyloc = this.lexer.yylloc;
                    if (recovering > 0) recovering--; /* las cosas van mejor si hubieron error */
                } else {
                    symbol = preErrorSymbol;
                    preErrorSymbol = null;
                }
                break;
            case 2:
                ...
                break;
            case 3:

```

```

        return true;
    }
}
return true;
}

```

4.10.9. Manejo de Errores

```

while (true) {
    state = stack[stack.length - 1];
    if (this.defaultActions[state]) { action = this.defaultActions[state]; }
    else {
        if (symbol === null || typeof symbol == "undefined") { symbol = lex(); }
        action = table[state] && table[state][symbol];
    }
    if (typeof action === "undefined" || !action.length || !action[0]) {
        var errStr = "";
        if (!recovering) { /* recovering = en estado de recuperación de un error */
            expected = []; /* computemos los tokens esperados */
            for (p in table[state]) /* si el estado "state" transita con p */
                if (this.terminals_[p] && p > 2) { /* y "p" es un terminal no especial */
                    expected.push("'" + this.terminals_[p] + "'"); /* entonces es esperado */
                }
            if (this.lexer.showPosition) { /* si esta definida la función showPosition */
                errStr = "Parse error on line " + (yylineno + 1) +
                    ":\n" + this.lexer.showPosition() +
                    "\nExpecting " + expected.join(", ") +
                    ", got '" +
                    (this.terminals_[symbol] || symbol) + /* terminals_ es el array invertido */
                    "'"; /* numero -> terminal
            }
        } else { /* ¡monta la cadena como puedas! */
            errStr = "Parse error on line " + (yylineno + 1) +
                ": Unexpected " +
                (symbol == 1 ? "end of input" : "'" +
                (this.terminals_[symbol] || symbol) + "'");
        }
        this.parseError(errStr, { /* genera la excepción */
            text: this.lexer.match, /* hash/objeto conteniendo los detalles del */
            token: this.terminals_[symbol] || symbol, /* error */
            line: this.lexer.yylineno,
            loc: yyloc,
            expected: expected
        });
    }
}
if (action[0] instanceof Array && action.length > 1) {
    throw new Error("Parse Error: multiple actions possible at state: " + state + ", token: " + symbol);
}
...
}

```

La función `parseError` genera una excepción:

```

parseError: function parseError(str, hash) {

```

```

        throw new Error(str); /* El hash contiene info sobre el error: token, linea, etc.
    },

```

- `parseError` es llamada cada vez que ocurre un error sintáctico. `str` contiene la cadena con el mensaje de error del tipo: `Expecting something, got other thing`. `hash` contiene atributos como `expected`: el array de tokens esperados; `line` la línea implicada, `loc` una descripción de la localización detallada del punto/terminal en el que ocurre el error; etc.

4.10.10. Analizador Léxico

El analizador léxico:

```

/* generated by jison-lex 0.1.0 */
var lexer = (function() {
    var lexer = {
        EOF: 1,
        parseError: function parseError(str, hash) { /* manejo de errores léxicos */ },
        setInput: function(input) { /* inicializar la entrada para el analizadorléxico */},
        input: function() { /* ... */ },
        unput: function(ch) { /* devolver al flujo de entrada */ },
        more: function() { /* ... */ },
        less: function(n) { /* ... */ },
        pastInput: function() { /* ... */ },
        upcomingInput: function() { /* ... */ },
        showPosition: function() { /* ... */ },
        next: function() {
            if (this.done) { return this.EOF; }
            if (!this._input) this.done = true;

            var token, match, tempMatch, index, col, lines;
            if (!this._more) { this.yytext = ''; this.match = ''; }
            var rules = this._currentRules();
            for (var i = 0; i < rules.length; i++) {
                tempMatch = this._input.match(this.rules[rules[i]]);
                if (tempMatch && (!match || tempMatch[0].length > match[0].length)) {
                    match = tempMatch;
                    index = i;
                    if (!this.options.flex) break;
                }
            }
            if (match) {
                lines = match[0].match(/(?:\r\n?|\n).*/g);
                if (lines) this.yylineno += lines.length;
                this.yylloc = {
                    first_line: this.yylloc.last_line,
                    last_line: this.yylineno + 1,
                    first_column: this.yylloc.last_column,
                    last_column:
                        lines ? lines[lines.length - 1].length -
                            lines[lines.length - 1].match(/\r?\n?/)[0].length
                        :
                            this.yylloc.last_column + match[0].length
                };
                this.yytext += match[0];

```



```

        this.match += match[0];
        this.matches = match;
        this.yyleng = this.yytext.length;
        if (this.options.ranges) {
            this.yylloc.range = [this.offset, this.offset += this.yyleng];
        }
        this._more = false;
        this._input = this._input.slice(match[0].length);
        this.matched += match[0];
        token = this.performAction.call(
            this,
            this.yy,
            this,
            rules[index],
            this.conditionStack[this.conditionStack.length - 1]
        );
        if (this.done && this._input) this.done = false;
        if (token) return token;
        else return;
    }
    if (this._input === "") { return this.EOF; }
    else {
        return this.parseError(
            'Lexical error on line ' + (this.yylineno + 1) +
            '. Unrecognized text.\n' + this.showPosition(),
            { text: "", token: null, line: this.yylineno }
        );
    }
},
lex: function lex() {
    var r = this.next();
    if (typeof r !== 'undefined') {
        return r;
    } else {
        return this.lex();
    }
},
begin: function begin(condition) { },
popState: function popState() { },
_currentRules: function _currentRules() { },
topState: function() { },
pushState: function begin(condition) { },
options: {},
performAction: function anonymous(yy, yy_, $avoiding_name_collisions, YY_START)
{
    var YYSTATE = YY_START;
    switch ($avoiding_name_collisions) {
        case 0:
            return yy_.yytext;
            break;
    }
},
rules: [/^(?:.)/], /* lista de expresiones regulares */

```

```

        conditions: { /* ... */ }
    }
};

```

4.10.11. Exportación

Si no ha sido exportado ya ...

```

if (typeof require !== 'undefined' && typeof exports !== 'undefined') {
    exports.parser = aSb;          /* hacemos accesible el objeto aSb */
    exports.Parser = aSb.Parser;
}

```

El objeto `aSb.Parser` representa al parser. Este es el código que lo crea.

```

function Parser() {
    this.yy = {};
}
Parser.prototype = parser;
parser.Parser = Parser;
return new Parser;
})();

```

También se exporta una función `parse`:

```

exports.parse = function() {
    return aSb.parse.apply(aSb, arguments);
};

```

y una función `main`:

```

exports.main = function commonjsMain(args) {
    if (!args[1]) {
        console.log('Usage: ' + args[0] + ' FILE');
        process.exit(1);
    }
    var source = require('fs').readFileSync(require('path').normalize(args[1]), "utf8");
    return exports.parser.parse(source);
};
if (typeof module !== 'undefined' && require.main === module) {
    exports.main(process.argv.slice(1));
}
}

```

Esto permite ejecutar el módulo directamente:

```

[~/Dropbox/src/javascript/PLgrado/jison-aSb(develop)]$ node aSb.js input.ab
empty
S -> aSb
S -> aSb
[~/Dropbox/src/javascript/PLgrado/jison-aSb(develop)]$ cat input.ab
aabb

```

```

~/Dropbox/src/javascript/PLgrado/jison-aSb(develop)]$ node debug aSb.js input.ab
< debugger listening on port 5858
connecting... ok
break in aSb.js:2
  1 /* parser generated by jison 0.4.2 */

```

```

    2 var aSb = (function() {
    3     var parser = {
    4         trace: function trace() {},
debug> n
break in aSb.js:390
    388     return new Parser;
    389 })();
    390 if (typeof require !== 'undefined' && typeof exports !== 'undefined') {
    391     exports.parser = aSb;
    392     exports.Parser = aSb.Parser;

debug> repl
Press Ctrl + C to leave debug repl
>
> typeof require
'function'
> typeof exports
'object'
> aSb
{ yy: {} }
> aSb.Parser
[Function]
^C
debug> sb(396)
    395     };
debug> c
break in aSb.js:396
    394     return aSb.parse.apply(aSb, arguments);
    395     };
*396     exports.main = function commonjsMain(args) {
    397         if (!args[1]) {
    398             console.log('Usage: ' + args[0] + ' FILE');
debug> n
break in aSb.js:404
    402     return exports.parser.parse(source);
    403     };
    404     if (typeof module !== 'undefined' && require.main === module) {
    405         exports.main(process.argv.slice(1));
    406     }
debug> repl
Press Ctrl + C to leave debug repl
> process.argv.slice(1)
[ '/Users/casiano/Dropbox/src/javascript/PLgrado/jison-aSb/aSb.js',
  'input.ab' ]
> typeof module
'object'
> require.main
{ id: '.',
  exports:
    { parser: { yy: {} },
      Parser: [Function],
      parse: [Function],
      main: [Function] },
  parent: null,

```

```

filename: '/Users/casiano/Dropbox/src/javascript/PLgrado/jison-aSb/aSb.js',
loaded: false,
children: [],
paths:
  [ '/Users/casiano/Dropbox/src/javascript/PLgrado/jison-aSb/node_modules',
    '/Users/casiano/Dropbox/src/javascript/PLgrado/node_modules',
    '/Users/casiano/Dropbox/src/javascript/node_modules',
    '/Users/casiano/Dropbox/src/node_modules',
    '/Users/casiano/Dropbox/node_modules',
    '/Users/casiano/node_modules',
    '/Users/node_modules',
    '/node_modules' ] }
^C
debug> n
break in aSb.js:405
403     };
404     if (typeof module !== 'undefined' && require.main === module) {
405         exports.main(process.argv.slice(1));
406     }
407 }
debug> n
< empty
< S -> aSb
< S -> aSb
break in aSb.js:409
407 }
408
409 });
debug> c
program terminated
debug>

```

4.11. Precedencia y Asociatividad

Recordemos que si al construir la tabla LALR, alguna de las entradas de la tabla resulta multievaluada, decimos que existe un conflicto. Si una de las acciones es de ‘reducción’ y la otra es de ‘desplazamiento’, se dice que hay un *conflicto shift-reduce* o *conflicto de desplazamiento-reducción*. Si las dos reglas indican una acción de reducción, decimos que tenemos un *conflicto reduce-reduce* o de *reducción-reducción*. En caso de que no existan indicaciones específicas *jison* resuelve los conflictos que aparecen en la construcción de la tabla utilizando las siguientes reglas:

1. Un conflicto *reduce-reduce* se resuelve eligiendo la producción que se listó primero en la especificación de la gramática.
2. Un conflicto *shift-reduce* se resuelve siempre en favor del *shift*

Las declaraciones de precedencia y asociatividad mediante las palabras reservadas `%left`, `%right`, `%nonassoc` se utilizan para modificar estos criterios por defecto. La declaración de `tokens` mediante la palabra reservada `%token` no modifica la precedencia. Si lo hacen las declaraciones realizadas usando las palabras `left`, `right` y `nonassoc`.

1. Los *tokens* declarados en la misma línea tienen igual precedencia e igual asociatividad. La precedencia es mayor cuanto mas abajo su posición en el texto. Así, en el ejemplo de la calculadora en la sección ??, el *token* `*` tiene mayor precedencia que `+` pero la misma que `/`.

2. La precedencia de una regla $A \rightarrow \alpha$ se define como la del terminal mas a la derecha que aparece en α . En el ejemplo, la producción

`expr : expr '+' expr`

tiene la precedencia del *token* `+`.

3. Para decidir en un conflicto *shift-reduce* se comparan la precedencia de la regla con la del terminal que va a ser desplazado. Si la de la regla es mayor se reduce si la del *token* es mayor, se desplaza.
4. Si en un conflicto *shift-reduce* ambos la regla y el terminal que va a ser desplazado tiene la misma precedencia *jison* considera la asociatividad, si es asociativa a izquierdas, reduce y si es asociativa a derechas desplaza. Si no es asociativa, genera un mensaje de error.
Obsérvese que, en esta situación, la asociatividad de la regla y la del *token* han de ser por fuerza, las mismas. Ello es así, porque en *jison* los *tokens* con la misma precedencia se declaran en la misma línea y sólo se permite una declaración por línea.
5. *Por tanto es imposible declarar dos tokens con diferente asociatividad y la misma precedencia.*
6. Es posible modificar la precedencia “natural” de una regla, calificándola con un *token* específico. para ello se escribe a la derecha de la regla `prec token`, donde `token` es un *token* con la precedencia que deseamos. Vea el uso del *token dummy* en el siguiente ejercicio.

Para ilustrar las reglas anteriores usaremos el siguiente programa *jison*:

```
[~/jison/jison-prec(ast)]$ cat -n precedencia.jison
1  %token NUMBER
2  %left '@'
3  %right '&' dummy
4  %%
5  s
6      : list      { console.log($list); }
7      ;
8
9  list
10     :
11         {
12             $$ = [];
13         }
14     | list '\n'
15         {
16             $$ = $1;
17         }
18     | list e
19         {
20             $$ = $1;
21             $$ .push($e);
22         }
23     ;
24
25 e : NUMBER
26     {
27         $$ = "NUMBER (" + yytext + ")";
28     }
29 | e '&' e
```

```

30         {
31             $$ = [ "&", $e1, $e2];
32         }
33     | e '@' e %prec dummy
34         {
35             $$ = ["@", $e1, $e2];
36         }
37     ;
38
39 %%

```

Obsérvese la siguiente ejecución:

```

[~/jison/jison-prec(ast)]$ cat input.txt
2@3@4
2&3&4
[~/jison/jison-prec(ast)]$ node precedencia.js input.txt
[ [ '@', [ '@', 'NUMBER (2)', 'NUMBER (3)' ], 'NUMBER (4)' ],
  [ '&', 'NUMBER (2)', [ '&', 'NUMBER (3)', 'NUMBER (4)' ] ] ]

```

Compilamos a continuación con bison usando la opción `-v` para producir información sobre los conflictos y las tablas de salto y de acciones:

```

[~/jison/jison-prec(ast)]$ bison -v precedencia.jison
precedencia.jison:6.31: warning: stray '$'
precedencia.jison:21.27: warning: stray '$'
precedencia.jison:31.31: warning: stray '$'
precedencia.jison:31.36: warning: stray '$'
precedencia.jison:35.30: warning: stray '$'
precedencia.jison:35.35: warning: stray '$'

```

La opción `-v` genera el fichero `Precedencia.output` el cual contiene información detallada sobre el autómata:

```

[~/jison/jison-prec(ast)]$ cat precedencia.output
Grammar

    0 $accept: s $end

    1 s: list

    2 list: /* empty */
    3     | list '\n'
    4     | list e

    5 e: NUMBER
    6   | e '&' e
    7   | e '@' e

```

Terminals, with rules where they appear

```

$end (0) 0
'\n' (10) 3
'&' (38) 6

```

'@' (64) 7
error (256)
NUMBER (258) 5
dummy (259)

Nonterminals, with rules where they appear

\$accept (8)
 on left: 0
s (9)
 on left: 1, on right: 0
list (10)
 on left: 2 3 4, on right: 1 3 4
e (11)
 on left: 5 6 7, on right: 4 6 7

state 0

0 \$accept: . s \$end

\$default reduce using rule 2 (list)

s go to state 1
list go to state 2

state 1

0 \$accept: s . \$end

\$end shift, and go to state 3

state 2

1 s: list .
3 list: list . '\n'
4 | list . e

NUMBER shift, and go to state 4
'\n' shift, and go to state 5

\$default reduce using rule 1 (s)

e go to state 6

state 3

0 \$accept: s \$end .

\$default accept

state 4

5 e: NUMBER .

\$default reduce using rule 5 (e)

state 5

3 list: list '\n' .

\$default reduce using rule 3 (list)

state 6

4 list: list e .

6 e: e . '&' e

7 | e . '@' e

'@' shift, and go to state 7

'&' shift, and go to state 8

\$default reduce using rule 4 (list)

state 7

7 e: e '@' . e

NUMBER shift, and go to state 4

e go to state 9

state 8

6 e: e '&' . e

NUMBER shift, and go to state 4

e go to state 10

state 9

6 e: e . '&' e

7 | e . '@' e

7 | e '@' e .


```
'&' shift, and go to state 8
```

```
$default reduce using rule 7 (e)
```

```
state 10
```

```
6 e: e . '&' e
```

```
6 | e '&' e .
```

```
7 | e . '@' e
```

```
'&' shift, and go to state 8
```

```
$default reduce using rule 6 (e)
```

La presencia de conflictos, aunque no siempre, en muchos casos es debida a la introducción de ambigüedad en la gramática. Si el conflicto es de desplazamiento-reducción se puede resolver explicitando alguna regla que rompa la ambigüedad. Los conflictos de reducción-reducción suelen producirse por un diseño erróneo de la gramática. En tales casos, suele ser mas adecuado modificar la gramática.

4.12. Esquemas de Traducción

Un *esquema de traducción* es una gramática independiente del contexto en la cuál se han asociado atributos a los símbolos de la gramática. Un atributo queda caracterizado por un identificador o nombre y un tipo o clase. Además se han insertado acciones, esto es, código JavaScript/Perl/Python/C, ... en medio de las partes derechas. En ese código es posible referenciar los atributos de los símbolos de la gramática como variables del lenguaje subyacente.

Recuerde que el orden en que se evalúan los fragmentos de código es el de un recorrido primero-profundo del árbol de análisis sintáctico. Mas específicamente, considerando a las acciones como hijos-hoja del nodo, el recorrido que realiza un esquema de traducción es:

```
1 function esquema_de_traduccion(node) {
2
3     for(c in node.children) { # de izquierda a derecha
4         child = node.children[i];
5         if (child instanceof 'SemanticAction') { # si es una acción semántica
6             child.execute;
7         }
8         else { esquema_de_traduccion(child) }
9     }
10 }
```

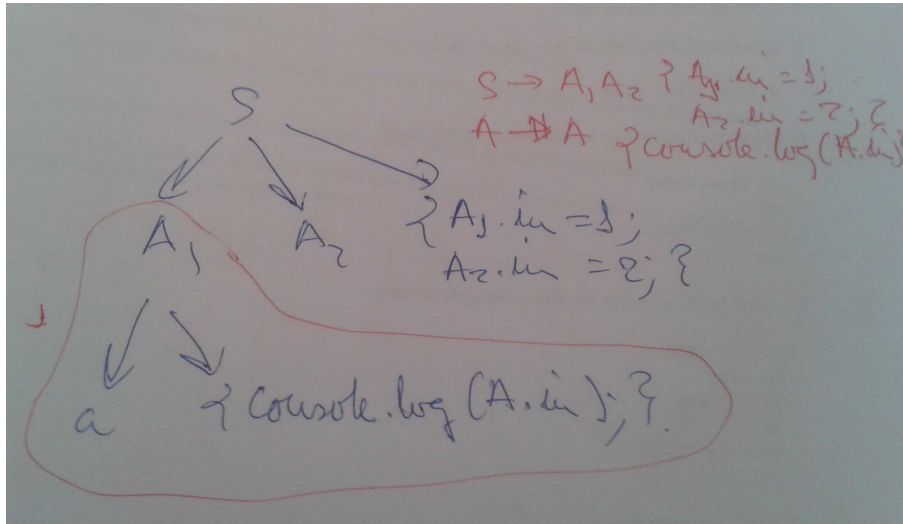
Obsérvese que, como el bucle recorre a los hijos de izquierda a derecha, se debe dar la siguiente condición para que un esquema de traducción funcione:

Para cualquier regla de producción aumentada con acciones, de la forma

$$A \rightarrow X_1 \dots X_j \{ \text{action}(A\{b\}, X_1\{c\} \dots X_n\{d\}) \} X_{j+1} \dots X_n$$

debe ocurrir que los atributos evaluados en la acción insertada después de X_j dependan de atributos y variables que fueron computadas durante la visita de los hermanos izquierdos o de sus ancestros. En particular no deberían depender de atributos asociados con las variables $X_{j+1} \dots X_n$. Ello no significa que no sea correcto evaluar atributos de $X_{j+1} \dots X_n$ en esa acción.

Por ejemplo, el siguiente esquema no satisface el requisito:



porque cuando vas a ejecutar la acción `{ console.log(A.in) }` el atributo `A.in` no ha sido computado.

Los atributos de cada símbolo de la gramática $X \in V \cup \Sigma$ se dividen en dos grupos disjuntos: *atributos sintetizados* y *atributos heredados*:

- Un atributo de X es un *atributo heredado* si depende de atributos de su padre y hermanos en el árbol.
- Un *atributo sintetizado* es aquél tal que el valor del atributo depende de los valores de los atributos de los hijos, es decir en tal caso X ha de ser una variable sintáctica y los atributos en la parte derecha de la regla semántica deben ser atributos de símbolos en la parte derecha de la regla de producción asociada.

4.13. Manejo en jison de Atributos Heredados

Supongamos que `jison` esta inmerso en la construcción de la antiderivación a derechas y que la forma sentencial derecha en ese momento es:

$$X_m \dots X_1 X_0 Y_1 \dots Y_n a_1 \dots a_0$$

y que el mango es $B \rightarrow Y_1 \dots Y_n$ y en la entrada quedan por procesar $a_1 \dots a_0$.

No es posible acceder en `jison` a los valores de los atributos de los estados en la pila del analizador que se encuentran “por debajo” o si se quiere “a la izquierda” de los estados asociados con la regla por la que se reduce.

Vamos a usar un pequeño hack para acceder a los atributos asociados con símbolos vistos en el pasado remoto”:

```
[~/jison/jison-inherited(grammar)]$ cat inherited.jison
%lex
%%
\s+          {}
(global|local|integer|float) { return yytext; }
[a-zA-Z_]\w* { return 'id'; }
.            { return yytext; }
/lex
%%
D
: C T L
```

```

;

C
: global
| local
;

T
: integer
| float
;

L
: L ',' id {
    console.log("L -> L ',' id (" + yytext + ")");
    var s = eval('$$');
    console.log(s);
}
| id {
    console.log("L -> id (" + yytext + ")");
    var s = eval('$$');
    console.log(s);
}

;
%%

```

Veamos un ejemplo de ejecución:

```

[~/jison/jison-inherited(grammar)]$ cat input.txt
global integer a, b, c
[~/jison/jison-inherited(grammar)]$ node inherited.js input.txt
L -> id (a)
[ null, 'global', 'integer', 'a' ]
L -> L ',' id (b)
[ null, 'global', 'integer', 'a', ',', 'b' ]
L -> L ',' id (c)
[ null, 'global', 'integer', 'a', ',', 'c' ]

```

Esta forma de acceder a los atributos es especialmente útil cuando se trabaja con *atributos heredados*. Esto es, cuando un atributo de un nodo del árbol sintáctico se computa en términos de valores de atributos de su padre y/o sus hermanos. Ejemplos de atributos heredados son la clase y tipo en la declaración de variables.

Es importante darse cuenta que en cualquier derivación a derechas desde D , cuando se reduce por una de las reglas

$$L \rightarrow id \mid L_1 ',' id$$

el símbolo a la izquierda de L es T y el que esta a la izquierda de T es C . Considere, por ejemplo la derivación a derechas:

$$\begin{aligned}
 D &\Rightarrow C T L \Rightarrow C T L, id \Rightarrow C T L, id, id \Rightarrow C T id, id, id \Rightarrow \\
 &\Rightarrow C float id, id, id \Rightarrow local float id, id, id
 \end{aligned}$$

Observe que el orden de recorrido de `jison` es:

$$\begin{aligned} &\text{local float id, id, id} \Leftarrow \text{C float id, id} \Leftarrow \text{C T id, id, id} \Leftarrow \\ &\quad \Leftarrow \text{C T L, id, id} \Leftarrow \text{C T L, id} \Leftarrow \text{C T L} \Leftarrow \text{D} \end{aligned}$$

en la antiderivación, cuando el mango es una de las dos reglas para listas de identificadores, $L \rightarrow \text{id}$ y $L \rightarrow L, \text{id}$ es decir durante las tres ultimas antiderivaciones:

$$\text{C T L, id, id} \Leftarrow \text{C T L, id} \Leftarrow \text{C T L} \Leftarrow \text{D}$$

las variables a la izquierda del mango son T y C. Esto ocurre siempre. Estas observaciones nos conducen al siguiente programa jison:

```
[~/jison/jison-inherited(deepstack)]$ cat inherited.jison
%lex
%%
\s+          {}
(global|local|integer|float) { return yytext; }
[a-zA-Z_]\w* { return 'id'; }
.             { return yytext; }
/lex
%%
D
  : C T L
  ;

C
  : global
  | local
  ;

T
  : integer
  | float
  ;

L
  : L ',' id {
    var s = eval('$$');
    var b0 = s.length - 3;

    console.log("L -> L ',' id (" + yytext + ")");
    console.log($id + ' is of type ' + s[b0-1]);
    console.log(s[b0] + ' is of class ' + s[b0-2]);
  }
  | id {
    var s = eval('$$');
    var b0 = s.length - 1;

    console.log("L -> id (" + yytext + ")");
    console.log($id + ' is of type ' + s[b0-1]);
    console.log(s[b0] + ' is of class ' + s[b0-2]);
  }
  ;
%%
```

A continuación sigue un ejemplo de ejecución:

```
[~/jison/jison-inherited(deepstack)]$ node inherited.js input.txt
L -> id (a)
a is of type integer
a is of class global
L -> L ',', id (b)
b is of type integer
a is of class global
L -> L ',', id (c)
c is of type integer
a is of class global
```

En este caso, existen varias alternativas simples a esta solución:

- Montar la lista de identificadores en un array y ponerles el tipo y la clase de "golpe".^{en} la regla de producción superior $D : C T L$;
- usar variables visibles (globales o atributos del objeto parser, por ejemplo) como `current_type`, `current_class`

```
C
: global { current_class = 'global'; }
| local { current_class = 'local'; }
```

y después accederlas en las reglas de L

- La que posiblemente sea la mejor estrategia general: construir el árbol de análisis sintáctico. Posteriormente podemos recorrer el árbol como queramos y tantas veces como queramos.

4.14. Definición Dirigida por la Sintáxis

Una *definición dirigida por la sintáxis* es un pariente cercano de los esquemas de traducción. En una definición dirigida por la sintáxis una gramática $G = (V, \Sigma, P, S)$ se aumenta con nuevas características:

- A cada símbolo $S \in V \cup \Sigma$ de la gramática se le asocian cero o mas atributos. Un atributo queda caracterizado por un identificador o nombre y un tipo o clase. A este nivel son *atributos formales*, como los parámetros formales, en el sentido de que su realización se produce cuando el nodo del árbol es creado.
- A cada regla de producción $A \rightarrow X_1 X_2 \dots X_n \in P$ se le asocian un conjunto de *reglas de evaluación de los atributos* o *reglas semánticas* que indican que el atributo en la parte izquierda de la regla semántica depende de los atributos que aparecen en la parte derecha de la regla. El atributo que aparece en la parte izquierda de la regla semántica puede estar asociado con un símbolo en la parte derecha de la regla de producción.
- Los atributos de cada símbolo de la gramática $X \in V \cup \Sigma$ se dividen en dos grupos disjuntos: *atributos sintetizados* y *atributos heredados*. Un atributo de X es un *atributo heredado* si depende de atributos de su padre y hermanos en el árbol. Un *atributo sintetizado* es aquél tal que el valor del atributo depende de los valores de los atributos de los hijos, es decir en tal caso X ha de ser una variable sintáctica y los atributos en la parte derecha de la regla semántica deben ser atributos de símbolos en la parte derecha de la regla de producción asociada.
- Los atributos predefinidos se denominán *atributos intrínsecos*. Ejemplos de atributos intrínsecos son los atributos sintetizados de los terminales, los cuáles se han computado durante la fase de análisis léxico. También son atributos intrínsecos los atributos heredados del símbolo de arranque, los cuales son pasados como parámetros al comienzo de la computación.

La diferencia principal con un esquema de traducción está en que no se especifica el orden de ejecución de las reglas semánticas. Se asume que, bien de forma manual o automática, se resolverán las dependencias existentes entre los atributos determinadas por la aplicación de las reglas semánticas, de manera que serán evaluados primero aquellos atributos que no dependen de ningún otro, después los que dependen de estos, etc. siguiendo un esquema de ejecución que viene guiado por las dependencias existentes entre los datos.

Aunque hay muchas formas de realizar un evaluador de una definición dirigida por la sintaxis, conceptualmente, tal evaluador debe:

1. Construir el árbol de análisis sintáctico para la gramática y la entrada dadas.
2. Analizar las reglas semánticas para determinar los atributos, su clase y las dependencias entre los mismos.
3. Construir el *grafo de dependencias* de los atributos, el cual tiene un nodo para cada ocurrencia de un atributo en el árbol de análisis sintáctico etiquetado con dicho atributo. El grafo tiene una arista entre dos nodos si existe una dependencia entre los dos atributos a través de alguna regla semántica.
4. Supuesto que el grafo de dependencias determina un *orden parcial* (esto es cumple las propiedades reflexiva, antisimétrica y transitiva) construir un *orden topológico* compatible con el orden parcial.
5. Evaluar las reglas semánticas de acuerdo con el orden topológico.

Una definición dirigida por la sintaxis en la que las reglas semánticas no tienen efectos laterales se denomina una *gramática atribuída*.

Si la definición dirigida por la sintaxis puede ser realizada mediante un esquema de traducción se dice que es *L-atribuída*. Para que una definición dirigida por la sintaxis sea L-atribuída deben cumplirse que cualquiera que sea la regla de producción $A \rightarrow X_1 \dots X_n$, los atributos heredados de X_j pueden depender únicamente de:

1. Los atributos de los símbolos a la izquierda de X_j
2. Los atributos heredados de A

Nótese que las restricciones se refieren a los atributos heredados. El cálculo de los atributos sintetizados no supone problema para un esquema de traducción. Si la gramática es LL(1), resulta fácil realizar una definición L-atribuída en un analizador descendente recursivo predictivo.

Si la definición dirigida por la sintaxis sólo utiliza atributos sintetizados se denomina *S-atribuída*. Una definición S-atribuída puede ser fácilmente trasladada a un programa `jison`.

Ejercicio 4.14.1. *El siguiente es un ejemplo de definición dirigida por la sintaxis:*

$S \rightarrow a A C$	$\$C\{i\} = \$A\{s\}$
$S \rightarrow b A B C$	$\$C\{i\} = \$A\{s\}$
$C \rightarrow c$	$\$C\{s\} = \$C\{i\}$
$A \rightarrow a$	$\$A\{s\} = "a"$
$B \rightarrow b$	$\$B\{s\} = "b"$

Determine un orden correcto de evaluación de la anterior definición dirigida por la sintaxis para la entrada `b a b c`.

Ejercicio 4.14.2.

*Lea el artículo *Are Attribute Grammars used in Industry?* por Josef Grosch*

I am observing a lack of education and knowledge about compiler construction in industry. When I am asking the participants of our trainings or the employees we met in our projects then only few people have learned about compiler construction during their education. For many of them compiler construction has a bad reputation because of what and how they have learned about this topic. Even fewer people have a usable knowledge about compilers. Even fewer people know about the theory of attribute grammars. And even fewer people know how to use attribute grammars for solving practical problems. Nevertheless, attribute grammars are used in industry. However, in many cases the people in industry do not know about this fact. They are running prefabricated subsystems constructed by external companies such as ours. These subsystems are for example parsers which use attribute grammar technology.

4.15. Ejercicios: Casos de Estudio

Véase nuestro proyecto Grammar Repository en [GoogleCode](#).

4.15.1. Un mal diseño

Ejercicio 4.15.1. *This grammar*

```
%token D S

%%
p: ds ';' ss | ss ;
ds: D ';' ds
   | D
;
ss: S ';' ss | S ;
%%
```

illustrates a typical LALR conflict due to a bad grammar design.

- *Reescriba la gramática para que no existan conflictos*
- *Escriba las acciones semánticas necesarias para imprimir la lista de Ds seguida de la lista de Ss*

4.15.2. Gramática no LR(1)

La siguiente gramática no es LR(1).

```
[~/srcPLgrado/jison/jison-nolr]$ cat confusingsolvedppcr.y
%%
A:
    B 'c' 'd'
  | E 'c' 'f'
;
B:
    'x' 'y'
;
E:
    'x' 'y'
;
%%
```

Encuentre una gramática sin conflictos equivalente a la anterior.

4.15.3. Un Lenguaje Intrínsecamente Ambiguo

Ejercicio 4.15.2. *A context-free language is inherently ambiguous if all context-free grammars generating that language are ambiguous. While some context-free languages have both ambiguous and unambiguous grammars, there are context-free languages for which no unambiguous context-free grammar can exist. An example of an inherently ambiguous language is the set*

$$\{ a^n b^n c^m : n \geq 0, m \geq 0 \} \cup \{ a^n b^m c^m : n \geq 0, m \geq 0 \}$$

Esto es: Concatenaciones de repeticiones de as seguidas de repeticiones de bs y seguidas de repeticiones de cs donde el número de as es igual al número de bs o bien el número de bs es igual al número de cs.

- *Escriba una gramática que genere dicho lenguaje*
- *Escriba un programa Jison que reconozca este lenguaje.*

4.15.4. Conflicto reduce-reduce

La siguiente gramática presenta conflictos reduce-reduce:

Ejercicio 4.15.3. `[~/srcPLgrado/jison/jison-reducereduceconflict]$ cat reducereducereconflictPPCR2.y`
`%token ID`

`%%`

```
def:    param_spec return_spec ','  
        ;
```

```
param_spec:  
    type  
    | name_list ':' type  
    ;
```

```
return_spec:  
    type  
    | name ':' type  
    ;
```

```
type:  
    ID  
    ;
```

```
name:  
    ID  
    ;
```

```
name_list:  
    name  
    | name ',' name_list  
    ;
```

`%%`

Este es el diagnóstico de Jison:

```
~/srcPLgrado/jison/jison-reducereduceconflict]$ jison reducereducereconflictPPCR2.y  
Conflict in grammar: multiple actions possible when lookahead token is ID in state 5  
- reduce by rule: name -> ID  
- reduce by rule: type -> ID  
Conflict in grammar: multiple actions possible when lookahead token is : in state 5  
- reduce by rule: name -> ID  
- reduce by rule: type -> ID  
Conflict in grammar: multiple actions possible when lookahead token is , in state 5  
- reduce by rule: name -> ID
```


- reduce by rule: type -> ID

States with conflicts:

State 5

```
type -> ID . #lookaheads= ID : ,
name -> ID . #lookaheads= ID : ,
```

Encuentre una gramática equivalente a la anterior sin conflictos.

4.16. Recuperación de Errores

La recuperación de errores no parece estar implementada en Jison. véase

- la sección Error Recovery de la documentación
- Pullreq 5 - parser built-in grammar error recovery was completely broken

```
[~/srcPLgrado/jison/jison-aSb(error)]$ cat aSb.jison
%lex
%%
\s+          {}
[ab]         { return yytext; }
.            { return "INVALID"; }
/lex
%%
S: /* empty */ { $$ = ''; console.log("empty"); }
  | 'a' S 'b' { $$ = $1 + $2 + $3; console.log("S -> aSb"); }
  | 'a' S error
;
%%

    parse: function parse(input) {
        var self = this,
            stack = [0],
            vstack = [null], // semantic value stack
            lstack = [], // location stack
            ...
            recovering = 0,
            ERROR = 2,
            EOF = 1;
        while (true) {
            // retrieve state number from top of stack
            state = stack[stack.length - 1];

            ...

            // handle parse error
            _handle_error: if (typeof action === 'undefined' || !action.length || ! ...

            var errStr = '';
            if (!recovering) {
```

4.17. Depuración en jison

4.18. Construcción del Árbol Sintáctico

El siguiente ejemplo usa `jison` para construir el árbol sintáctico de una expresión en infijo:

4.19. Consejos a seguir al escribir un programa jison

Cuando escriba un programa `jison` asegúrese de seguir los siguientes consejos:

1. Coloque el punto y coma de separación de reglas en una línea aparte. Un punto y coma “pegado” al final de una regla puede confundirse con un terminal de la regla.
2. Si hay una regla que produce vacío, coloquela en primer lugar y acompáñela de un comentario resaltando ese hecho.
3. Nunca escriba dos reglas de producción en la misma línea.
4. Sangre convenientemente todas las partes derechas de las reglas de producción de una variable, de modo que queden alineadas.
5. Ponga nombres representativos a sus variables sintácticas. No llame `Z` a una variable que representa el concepto “lista de parámetros”, llámela `ListaDeParametros`.
6. Es conveniente que declare los terminales simbólicos, esto es, aquellos que llevan un identificador asociado. Si no llevan prioridad asociada o no es necesaria, use una declaración `%token`. De esta manera el lector de su programa se dará cuenta rápidamente que dichos identificadores no se corresponden con variables sintácticas. Por la misma razón, si se trata de terminales asociados con caracteres o cadenas no es tan necesario que los declare, a menos que, como en el ejemplo de la calculadora para `'+'` y `'*'`, sea necesario asociarles una precedencia.
7. Es importante que use la opción `-v` para producir el fichero `.output` conteniendo información detallada sobre los conflictos y el autómata. Cuando haya un conflicto `shift-reduce` no resuelto busque en el fichero el estado implicado y vea que `LR(0)` items $A \rightarrow \alpha\uparrow$ y $B \rightarrow \beta\uparrow\gamma$ entran en conflicto.
8. Si según el informe de `jison` el conflicto se produce ante un terminal a , es porque $a \in FOLLOW(A)$ y $a \in FIRST(\gamma)$. Busque las causas por las que esto ocurre y modifique su gramática con vistas a eliminar la presencia del terminal a en uno de los dos conjuntos implicados o bien establezca reglas de prioridad entre los terminales implicados que resuelvan el conflicto.
9. Nótese que cuando existe un conflicto de desplazamiento reducción entre $A \rightarrow \alpha\uparrow$ y $B \rightarrow \beta\uparrow\gamma$, el programa `jison` contabiliza un error por cada terminal $a \in FOLLOW(A) \cap FIRST(\gamma)$. Por esta razón, si hay 16 elementos en $FOLLOW(A) \cap FIRST(\gamma)$, el analizador `jison` informará de la existencia de 16 conflictos `shift-reduce`, cuando en realidad se trata de uno sólo. No desespere, los conflictos “auténticos” suelen ser menos de los que `jison` anuncia.
10. Si necesita declarar variables globales, inicializaciones, etc. que afectan la conducta global del analizador, escriba el código correspondiente en la cabecera del analizador, protegido por los delimitadores `%{` y `%}`. Estos delimitadores deberán aparecer en una línea aparte. Por ejemplo:

```
%{  
our contador = 0;  
%}  
  
%token NUM  
...  
%%
```

Capítulo 5

Análisis Sintáctico Ascendente en Ruby

5.1. La Calculadora

5.1.1. Uso desde Línea de Comandos

```
[~/src/PL/rexical/sample(master)]$ racc --help
Usage: racc [options] <input>
  -o, --output-file=PATH      output file name [<input>.tab.rb]
  -t, --debug                  Outputs debugging parser.
  -g                           Equivalent to -t (obsolete).
  -v, --verbose                Creates <filename>.output log file.
  -O, --log-file=PATH          Log file name [<input>.output]
  -e, --executable [RUBYPATH] Makes executable parser.
  -E, --embedded               Embeds Racc runtime in output.
      --line-convert-all       Converts line numbers of user codes.
  -l, --no-line-convert        Never convert line numbers.
  -a, --no-omit-actions        Never omit actions.
      --superclass=CLASSNAME    Uses CLASSNAME instead of Racc::Parser.
      --runtime=FEATURE         Uses FEATURE instead of 'racc/parser'
  -C, --check-only             Checks syntax and quit immediately.
  -S, --output-status           Outputs internal status time to time.
  -P                           Enables generator profile
  -D flags                     Flags for Racc debugging (do not use).
      --version                 Prints version and quit.
      --runtime-version         Prints runtime version and quit.
      --copyright               Prints copyright and quit.
      --help                    Prints this message and quit.
```

```
[~/Dropbox/src/PL/rexical/sample(master)]$ cat -n Rakefile
 1 task :default => %W{racc rex} do
 2   sh "ruby calc3.tab.rb"
 3 end
 4
 5 task :racc do
 6   sh "racc calc3.racc"
 7 end
 8
 9 task :rex do
10   sh "rex calc3.rex"
11 end
```

5.1.2. Análisis Léxico con rexical

```
[~/Dropbox/src/PL/rexical/sample(master)]$ cat -n calc3.rex
 1 #
 2 # calc3.rex
 3 # lexical scanner definition for rex
 4 #
 5
 6 class Calculator3
 7 macro
 8   BLANK      \s+
 9   DIGIT      \d+
10 rule
11   {BLANK}
12   {DIGIT}    { [:NUMBER, text.to_i] }
13   .|\n      { [text, text] }
14 inner
15 end
```

5.1.3. Análisis Sintáctico

```
[~/Dropbox/src/PL/rexical/sample(master)]$ cat -n calc3.racc
 1 #
 2 # A simple calculator, version 3.
 3 #
 4
 5 class Calculator3
 6   prehigh
 7     nonassoc UMINUS
 8     left '*' '/'
 9     left '+' '-'
10   preclow
11   options no_result_var
12 rule
13   target  : exp
14           | /* none */ { 0 }
15
16   exp     : exp '+' exp { val[0] + val[2] }
17           | exp '-' exp { val[0] - val[2] }
18           | exp '*' exp { val[0] * val[2] }
19           | exp '/' exp { val[0] / val[2] }
20           | '(' exp ')' { val[1] }
21           | '-' NUMBER =UMINUS { -(val[1]) }
22           | NUMBER
23 end
24
25 ---- header ----
26 #
27 # generated by racc
28 #
29 require 'calc3.rex'
30
31 ---- inner ----
32
```

```

33 ---- footer ----
34
35 puts 'sample calc'
36 puts '"q" to quit.'
37 calc = Calculator3.new
38 while true
39   print '>>> '; $stdout.flush
40   str = $stdin.gets.strip
41   break if /q/i === str
42   begin
43     p calc.scan_str(str)
44   rescue ParseError
45     puts 'parse error'
46   end
47 end

```

Precedencias right is yacc's %right, left is yacc's %left.
 = SYMBOL means yacc's %prec SYMBOL:

```

prehigh
  nonassoc '++'
  left     '*' '/'
  left     '+' '-'
  right    '='
preclow

rule
  exp: exp '*' exp
      | exp '-' exp
      | '-' exp    =UMINUS # equals to "%prec UMINUS"
      :
      :

```

Atributos You can use following special local variables in action.

1. **result (\$\$)**
 The value of left-hand side (lhs). A default value is `val[0]`.
2. **val (\$1,\$2,\$3...)**
 An array of value of right-hand side (rhs).
3. **_values (...\$-2,\$-1,\$0)**
 A stack of values. DO NOT MODIFY this stack unless you know what you are doing.

Declaring Tokens By declaring tokens, you can avoid bugs.

```
token NUM ID IF
```

Opciones

You can write options for yacc command in your yacc file.

```
options OPTION OPTION ...
```

Options are:

1. `omit_action_call`
omit empty action call or not.
2. `result_var`
use/does not use local variable result"

You can use `no_` prefix to invert its meanings.

User Code Block

User Code Block is a Ruby source code which is copied to output. There are three user code block, "header", "inner", and "footer".

Format of user code is like this:

```
---- header
  ruby statement
  ruby statement
  ruby statement

---- inner
  ruby statement
  :
  :
```

If four - exist on line head, racc treat it as beginning of user code block. A name of user code must be one word.

5.2. Véase También

- Racc en GitHub
-
- Racc User's Manual
- Martin Fowler Hello Racc
- Rexical en GitHub

Parte II

PARTE: CREATE YOUR OWN PROGRAMMING LANGUAGE

A course by Nathan Whitehead.

- Nathan Whitehead en YouTube

Repositorios relacionados:

- <https://github.com/crguezl/nathanuniversityexercisesPL>

Capítulo 6

JavaScript Review

<http://nathansuniversity.com/jsreview.html>

6.1. Closures

<http://nathansjslessons.appspot.com/>

Capítulo 7

Your First Compiler

<http://nathansuniversity.com/music.html>

Capítulo 8

Parsing

<http://nathansuniversity.com/pegs.html>

Capítulo 9

Scheem Interpreter

<http://nathansuniversity.com/scheem.html>

9.1. Scheem Interpreter

9.2. Variables

9.3. Setting Values

9.4. Putting Things Together

9.4.1. Unit Testing: Mocha

Introducción

Mocha is a feature-rich JavaScript test framework running on node.js and the browser, making asynchronous testing simple and fun. Mocha tests run serially, allowing for flexible and accurate reporting, while mapping uncaught exceptions to the correct test cases.

- <http://visionmedia.github.io/mocha/>
- <https://github.com/visionmedia/mocha>
- - An example setup for unit testing JavaScript in the browser with the Mocha testing framework and Chai assertions: <https://github.com/ludovicofischer/mocha-chai-browser-demo>
 - Karma - a test runner

mocha init

```
[~/srcPLgrado/mocha-chai-browser-demo(master)]$ mocha --help
```

```
Usage: _mocha [debug] [options] [files]
```

Commands:

```
init <path>           initialize a client-side mocha setup at <path>
```

Options:

-h, --help	output usage information
-V, --version	output the version number
-r, --require <name>	require the given module
-R, --reporter <name>	specify the reporter to use

-u, --ui <name>	specify user-interface (bdd tdd exports)
-g, --grep <pattern>	only run tests matching <pattern>
-i, --invert	inverts --grep matches
-t, --timeout <ms>	set test-case timeout in milliseconds [2000]
-s, --slow <ms>	"slow" test threshold in milliseconds [75]
-w, --watch	watch files for changes
-c, --colors	force enabling of colors
-C, --no-colors	force disabling of colors
-G, --growl	enable growl notification support
-d, --debug	enable node's debugger, synonym for node --debug
-b, --bail	bail after first test failure
-A, --async-only	force all tests to take a callback (async)
-S, --sort	sort test files
--recursive	include sub directories
--debug-brk	enable node's debugger breaking on the first line
--globals <names>	allow the given comma-delimited global [names]
--check-leaks	check for global variable leaks
--interfaces	display available interfaces
--reporters	display available reporters
--compilers <ext>:<module>,...	use the given module(s) to compile files
--inline-diffs	display actual/expected differences inline within each str
--no-exit	require a clean shutdown of the event loop: mocha will not

```
[~/srcPLgrado]$ mocha init chuchu
```

```
[~/srcPLgrado]$ ls -ltr
```

```
total 16
```

```
....
```

```
drwxr-xr-x  6 casiano  staff  204 20 ene 11:16 chuchu
```

```
[~/srcPLgrado]$ tree chuchu/
```

```
chuchu/
|-- index.html
|-- mocha.css
|-- mocha.js
'-- tests.js
```

```
[~/srcPLgrado/mocha-tutorial]$ cat test/test.js
```

```
var assert = require("assert")
describe('Array', function(){
  describe('#indexOf()', function(){
    it('should return -1 when the value is not present', function(){
      assert.equal(-1, [1,2,3].indexOf(5));
      assert.equal(-1, [1,2,3].indexOf(0));
      assert.equal( 0, [1,2,3].indexOf(99));
    })
  })
})
```

```
[~/srcPLgrado/mocha-tutorial]$ mocha
```

```
.
0 passing (5ms)
1 failing
```

```
1) Array #indexOf() should return -1 when the value is not present:
```

```
AssertionError: 0 == -1
```

```
at Context.<anonymous> (/Users/casiano/local/src/javascript/PLgrado/mocha-tutorial/test/
```

Mocha allows you to use any assertion library you want, if it throws an error, it will work! This means you can utilize libraries such as `should.js`, node's regular `assert` module, or others.

Browser support

Mocha runs in the browser.

- Every release of Mocha will have new builds of `./mocha.js` and `./mocha.css` for use in the browser.
- To setup Mocha for browser use all you have to do is include the script, stylesheet,
- Tell Mocha which interface you wish to use, and then
- Run the tests.

A typical setup might look something like the following, where we call `mocha.setup('bdd')` to use the BDD interface before loading the test scripts, running them on load with `mocha.run()`.

```
<html>
<head>
  <meta charset="utf-8">
  <title>Mocha Tests</title>
  <link rel="stylesheet" href="mocha.css" />
</head>
<body>
  <div id="mocha"></div>
  <script src="jquery.js"></script>
  <script src="expect.js"></script>
  <script src="mocha.js"></script>

  <script>mocha.setup('bdd')</script>

  <script src="test.array.js"></script>
  <script src="test.object.js"></script>
  <script src="test.xhr.js"></script>

  <script>
    mocha.checkLeaks();
    mocha.globals(['jQuery']);
    mocha.run();
  </script>
</body>
</html>
```

- Mocha "interface" system allows developers to choose their style of DSL. Shipping with BDD, TDD, and exports flavoured interfaces.
- `mocha.globals([names ...])`
A list of accepted global variable names. For example, suppose your app deliberately exposes a global named `app` and `YUI`
- `mocha.checkLeaks()`

By default Mocha will not check for global variables leaked while running tests

TDD

The *Mocha TDD interface* provides `suite()`, `test()`, `setup()`, and `teardown()`.

```
suite('Array', function(){
  setup(function(){
    // ...
  });

  suite('#indexOf()', function(){
    test('should return -1 when not present', function(){
      assert.equal(-1, [1,2,3].indexOf(4));
    });
  });
});
```

Véase

- <https://github.com/crguezl/nathanuniversityexercisesPL/tree/master/scheem8>

9.4.2. Karma

- *Karma* (See Karma installation) is essentially a tool which spawns a web server that executes source code against test code for each of the browsers connected.
- The results for each test against each browser are examined and displayed via the command line to the developer such that they can see which browsers and tests passed or failed.
- A browser can be captured either
 - manually, by visiting the URL where the Karma server is listening (typically `http://localhost:9876/`)
 - or automatically by letting Karma know which browsers to start when Karma is run
- Karma also watches all the files, specified within the configuration file, and whenever any file changes, it triggers the test run by sending a signal the testing server to inform all of the captured browsers to run the test code again.
- Each browser then loads the source files inside an `IFrame`¹, executes the tests and reports the results back to the server.
- The server collects the results from all of the captured browsers and presents them to the developer.
- JS.everywhere(Europe) 2012: Testacular, the Spectacular JavaScript Test Runner - Vojta Jína YouTube
- Google Test Automation Conference GTAC 2013: Karma - Test Runner for JavaScript Vojta Jína. YouTube

```
[~/srcPLgrado/mocha-chai-browser-demo(master)]$ karma --help
Karma - Spectacular Test Runner for JavaScript.
```

Usage:

```
/usr/local/bin/karma <command>
```

¹The `iframe` tag specifies an inline frame. An inline frame is used to embed another document within the current HTML document

Commands:

```
start [<configFile>] [<options>] Start the server / do single run.
init [<configFile>] Initialize a config file.
run [<options>] [ -- <clientArgs>] Trigger a test run.
completion Shell completion for karma.
```

Run `--help` with particular command to see its description and available options.

Options:

```
--help      Print usage and options.
--version   Print current version.
```

In order to serve us well, Karma needs to know about our project in order to test it and this is done via a configuration file.

The configuration file can be generated using `karma init`:

```
$ karma init my.conf.js
```

Which testing framework do you want to use ?

Press tab to list possible options. Enter to move to the next question.

```
> jasmine
```

Do you want to use Require.js ?

This will add Require.js plugin.

Press tab to list possible options. Enter to move to the next question.

```
> no
```

<http://requirejs.org/>

Do you want to capture a browser automatically ?

Press tab to list possible options. Enter empty string to move to the next question.

```
> Chrome
```

What is the location of your source and test files ?

You can use glob patterns, eg. `"js/*.js"` or `"test/**/*.Spec.js"`.

Enter empty string to move to the next question.

```
>
```

Should any of the files included by the previous patterns be excluded ?

You can use glob patterns, eg. `"**/*.swp"`.

Enter empty string to move to the next question.

Do you want Karma to watch all the files and run the tests on change ?

Press tab to list possible options.

```
> yes
```

Config file generated at `"/Users/casiano/local/src/javascript/PLgrado/mocha-tutorial/karma.conf.js"`

The configuration file can be written in CoffeeScript as well. In fact, if you execute `karma init` with a `.coffee` filename extension, it will generate a CoffeeScript file.

Of course, you can write the config file by hand or copy paste it from another project ;-)


```

[~/srcPLgrado/mocha-tutorial]$ cat karma.conf.js
// Karma configuration
// Generated on Mon Jan 20 2014 16:21:22 GMT+0000 (WET)

module.exports = function(config) {
  config.set({

    // base path, that will be used to resolve files and exclude
    basePath: '',

    // frameworks to use
    frameworks: ['jasmine'],

    // list of files / patterns to load in the browser
    files: [

    ],

    // list of files to exclude
    exclude: [

    ],

    // test results reporter to use
    // possible values: 'dots', 'progress', 'junit', 'growl', 'coverage'
    reporters: ['progress'],

    // web server port
    port: 9876,

    // enable / disable colors in the output (reporters and logs)
    colors: true,

    // level of logging
    // possible values: config.LOG_DISABLE || config.LOG_ERROR || config.LOG_WARN || config.LOG_INFO || config.LOG_DEBUG
    logLevel: config.LOG_INFO,

    // enable / disable watching file and executing tests whenever any file changes
    autoWatch: true,

    // Start these browsers, currently available:
    // - Chrome
    // - ChromeCanary
    // - Firefox

```

```

// - Opera (has to be installed with 'npm install karma-opera-launcher')
// - Safari (only Mac; has to be installed with 'npm install karma-safari-launcher')
// - PhantomJS
// - IE (only Windows; has to be installed with 'npm install karma-ie-launcher')
browsers: ['Chrome', 'Firefox'],

// If browser does not capture in given timeout [ms], kill it
captureTimeout: 60000,

// Continuous Integration mode
// if true, it capture browsers, run tests and exit
singleRun: false
});
};

```

When starting Karma, the configuration file path can be passed in as the first argument. By default, Karma will look for `karma.conf.js` in the current directory.

```

# Start Karma using your configuration
$ karma start my.conf.js

```

Some configurations, which are already present within the configuration file, can be overridden by specifying the configuration as a command line argument for when Karma is executed.

```
karma start karma-conf.js --command-one --command-two
```

```

[~/srcPLgrado/mocha-tutorial]$ karma start --help
Karma - Spectacular Test Runner for JavaScript.

```

START - Start the server / do a single run.

Usage:

```
/usr/local/bin/karma start [<configFile>] [<options>]
```

Options:

<code>--port</code>	<code><integer></code> Port where the server is running.
<code>--auto-watch</code>	Auto watch source files and run on change.
<code>--no-auto-watch</code>	Do not watch source files.
<code>--log-level</code>	<code><disable error warn info debug></code> Level of logging.
<code>--colors</code>	Use colors when reporting and printing logs.
<code>--no-colors</code>	Do not use colors when reporting or printing logs.
<code>--reporters</code>	List of reporters (available: dots, progress, junit, growl, coverage).
<code>--browsers</code>	List of browsers to start (eg. <code>--browsers Chrome,ChromeCanary,Firefox</code>).
<code>--capture-timeout</code>	<code><integer></code> Kill browser if does not capture in given time [ms].
<code>--single-run</code>	Run the test when browsers captured and exit.
<code>--no-single-run</code>	Disable single-run.
<code>--report-slower-than</code>	<code><integer></code> Report tests that are slower than given time [ms].
<code>--help</code>	Print usage and options.

Using Karma with Mocha To use Karma with Mocha we need the `karma-mocha` adapter.

If we want to pass configuration options directly to mocha you can do this in the following way

```
// karma.conf.js
```

```

module.exports = function(config) {
  config.set({
    frameworks: ['mocha'],

    files: [
      '*.js'
    ],

    client: {
      mocha: {
        ui: 'tdd'
      }
    }
  });
};

```

(By default the ui is bdd).

Here is an example (<https://github.com/crguezl/nathanuniversityexercisesPL/blob/master/scheem8/karma.conf.js>)

```

[~/srcPLgrado/nathansuniversity/exercises/scheem8(master)]$ cat karma.conf.js
// Karma configuration
// Generated on Tue Jan 21 2014 12:20:45 GMT+0000 (WET)

```

```

module.exports = function(config) {

  config.set({

    // base path, that will be used to resolve files and exclude
    basePath: '',

    // frameworks to use
    frameworks: ['mocha'],

    // list of files / patterns to load in the browser
    files: [
      'js/chai.js',
      'js/jquery-1.10.2.js',
      'js/mocha.js',
      'js/scheem8.js',
      'js/simpletest.js'
    ],

    // list of files to exclude
    exclude: [

    ],

    // test results reporter to use
    // possible values: 'dots', 'progress', 'junit', 'growl', 'coverage'
    reporters: ['progress'],
  });
};

```

```

// web server port
port: 9876,

// enable / disable colors in the output (reporters and logs)
colors: true,

// level of logging
// possible values: config.LOG_DISABLE || config.LOG_ERROR || config.LOG_WARN || config.LOG_INFO || config.LOG_DEBUG
logLevel: config.LOG_INFO,

// enable / disable watching file and executing tests whenever any file changes
autoWatch: true,

// Start these browsers, currently available:
// - Chrome
// - ChromeCanary
// - Firefox
// - Opera (has to be installed with 'npm install karma-opera-launcher')
// - Safari (only Mac; has to be installed with 'npm install karma-safari-launcher')
// - PhantomJS
// - IE (only Windows; has to be installed with 'npm install karma-ie-launcher')
browsers: ['Chrome', 'Firefox'],

// If browser does not capture in given timeout [ms], kill it
captureTimeout: 60000,

// Continuous Integration mode
// if true, it capture browsers, run tests and exit
singleRun: false,

client: {
  mocha: {
    ui: 'tdd'
  }
}
});
};

```

Load HTML files with Karma

If you have one html file:

```

[~/srcPLgrado/karma/html]$ cat template.html
<div id="tpl">content of the template</div>

```

which you want to load and then get all elements from that html page in your test script, you can use the `html2js` preprocessor, which basically converts HTML files into JavaScript strings and include these files.

```
[~/srcPLgrado/karma/html]$ cat karma.conf.js
module.exports = function(karma) {
  karma.configure({
    basePath: '',
    frameworks: ['jasmine'],
    files: [ '*.js', '*.html' ],
    preprocessors: { '*.html': 'html2js' },
    ....
  });
};
```

Then, you can access these strings in your test:

```
[~/srcPLgrado/karma/html]$ cat test.js
describe('template', function() {
  it('should expose the templates to __html__', function() {
    document.body.innerHTML = __html__['template.html'];
    expect(document.getElementById('tpl')).toBeDefined();
  });
});
```

See

- Load HTML files with Karma in StackOverflow.
- karma-html2js-preprocessor
- Example

9.4.3. Grunt

<http://gruntjs.com/getting-started>

```
npm install -g grunt-cli
```

A typical setup will involve adding two files to your project: **package.json** and the **Gruntfile**.

- **package.json**: This file is used by **npm** to store metadata for projects published as **npm** modules. You will list grunt and the Grunt plugins your project needs as *devDependencies* in this file.
- **Gruntfile**: This file is named **Gruntfile.js** or **Gruntfile.coffee** and is used to configure or define tasks and load Grunt plugins.

package.json

- The package.json file belongs in the root directory of your project, next to the Gruntfile, and should be committed with your project source.
- Running `npm install` in the same folder as a package.json file will install the correct version of each dependency listed therein.
- There are a few ways to create a package.json file for your project:
 - Most grunt-init templates will automatically create a project-specific package.json file.
 - The `npm init` command will create a basic package.json file.
 - Start with the example below, and expand as needed, following this specification.

```

{
  "name": "my-project-name",
  "version": "0.1.0",
  "devDependencies": {
    "grunt": "~0.4.2",
    "grunt-contrib-jshint": "~0.6.3",
    "grunt-contrib-nodeunit": "~0.2.0",
    "grunt-contrib-uglify": "~0.2.2"
  }
}

```

Gruntfile

The Gruntfile.js or Gruntfile.coffee file is a valid JavaScript or CoffeeScript file that belongs in the root directory of your project, next to the package.json file, and should be committed with your project source.

A Gruntfile is comprised of the following parts:

- The "wrapper" function
- Project and task configuration
- Loading Grunt plugins and tasks
- Custom tasks

An example Gruntfile

In the following Gruntfile, project metadata is imported into the Grunt config from the project's package.json file and the

grunt-contrib-uglify

plugin's uglify task is configured to minify a source file and generate a banner comment dynamically using that metadata.

When grunt is run on the command line, the uglify task will be run by default.

```

module.exports = function(grunt) {

  // Project configuration.
  grunt.initConfig({
    pkg: grunt.file.readJSON('package.json'),
    uglify: {
      options: {
        banner: '/*! <%= pkg.name %> <%= grunt.template.today("yyyy-mm-dd") %> */\n',
      },
      build: {
        src: 'src/<%= pkg.name %>.js',
        dest: 'build/<%= pkg.name %>.min.js'
      }
    }
  });

  // Load the plugin that provides the "uglify" task.
  grunt.loadNpmTasks('grunt-contrib-uglify');

  // Default task(s).

```

```

    grunt.registerTask('default', ['uglify']);
};

```

Now that you've seen the whole Gruntfile, let's look at its component parts.

The "wrapper" function

Every Gruntfile (and gruntplugin) uses this basic format, and all of your Grunt code must be specified inside this function:

```

module.exports = function(grunt) {
    // Do grunt-related things in here
};

```

Project and task configuration

Most Grunt tasks rely on configuration data defined in an object passed to the `grunt.initConfig` method.

In this example, `grunt.file.readJSON('package.json')` imports the JSON metadata stored in `package.json` into the grunt config. Because `<% %>` template strings may reference any config properties, configuration data like filepaths and file lists may be specified this way to reduce repetition.

You may store any arbitrary data inside of the configuration object, and as long as it doesn't conflict with properties your tasks require, it will be otherwise ignored. Also, because this is JavaScript, you're not limited to JSON; you may use any valid JS here. You can even programmatically generate the configuration if necessary.

Like most tasks, the `grunt-contrib-uglify` plugin's `uglify` task expects its configuration to be specified in a property of the same name. Here, the `banner` option is specified, along with a single `uglify` target named `build` that minifies a single source file to a single destination file.

```

// Project configuration.
grunt.initConfig({
  pkg: grunt.file.readJSON('package.json'),
  uglify: {
    options: {
      banner: '/*! <%= pkg.name %> <%= grunt.template.today("yyyy-mm-dd") %> */\n'
    },
    build: {
      src: 'src/<%= pkg.name %>.js',
      dest: 'build/<%= pkg.name %>.min.js'
    }
  }
});

```

A simple Grunt.js example

<https://github.com/UWMadisonUcomm/grunt-simple-example>

```

[~/srcPLgrado/grunt-simple-example(master)]$ pwd
/Users/casiano/srcPLgrado/grunt-simple-example
[~/srcPLgrado/grunt-simple-example(master)]$ git remote -v
origin  git@github.com:UWMadisonUcomm/grunt-simple-example.git (fetch)
origin  git@github.com:UWMadisonUcomm/grunt-simple-example.git (push)
[~/srcPLgrado/grunt-simple-example(master)]$ ls
Gruntfile.js  README.md  assets  index.html  node_modules  package.json  src

```

```
[~/srcPLgrado/grunt-simple-example(master)]$ cat Gruntfile.js
module.exports = function(grunt){
  grunt.initConfig({
    uglify: {
      main: {
        files: {
          'assets/app.min.js': [
            'src/javascripts/jquery-1.10.2.min.js',
            'src/javascripts/bootstrap.js',
            'src/javascripts/application.js'
          ]
        }
      }
    },
    less: {
      application: {
        options: {
          yuicompress: true
        },
        files: {
          "assets/app.min.css": "src/stylesheets/application.less"
        }
      }
    },
    watch: {
      javascripts: {
        files: ['src/javascripts/**/*'],
        tasks: ['uglify']
      },
      stylesheets: {
        files: ['src/stylesheets/**/*'],
        tasks: ['less']
      }
    }
  });

  // Load plugins
  grunt.loadNpmTasks('grunt-contrib-less');
  grunt.loadNpmTasks('grunt-contrib-uglify');
  grunt.loadNpmTasks('grunt-contrib-watch');

  // Register tasks
  grunt.registerTask('default', ['uglify', 'less']);
}
```

```
[~/srcPLgrado/grunt-simple-example(master)]$ cat package.json
{
  "name": "grunt-simple-example",
  "version": "0.0.1",
  "main": "index.js",
  "devDependencies": {
    "grunt": "~0.4.1",
    "grunt-contrib-cssmin": "~0.6.2",
    "grunt-contrib-less": "~0.7.0",
```



```

    "grunt-contrib-uglify": "~0.2.4",
    "grunt-contrib-watch": "~0.5.3"
  },
  "author": "Bryan Shelton",
  "license": "BSD-2-Clause"
}

```

```

[~/srcPLgrado/grunt-simple-example(master)]$ npm install
npm WARN package.json grunt-simple-example@0.0.1 No repository field.
[~/srcPLgrado/grunt-simple-example(master)]$

```

```

[~/srcPLgrado/grunt-simple-example(master)]$ grunt watch
Running "watch" task
Waiting...OK
>> File "src/javascripts/application.js" changed.

```

```

Running "uglify:main" (uglify) task
File "assets/app.min.js" created.

```

Done, without errors.

Completed in 3.897s at Mon Jan 20 2014 19:02:03 GMT+0000 (WET) - Waiting...

9.4.4. GitHub Project Pages

Project Pages are kept in the same repository as the project they are for.

These pages are similar to User and Org Pages, with a few slight differences:

- The `gh-pages` branch is used to build and publish from.
- A custom domain on user/org pages will apply the same domain redirect to all project pages hosted under that account, unless the project pages use their own custom domain.
- If no custom domain is used, the project pages are served under a subpath of the user pages:

`username.github.io/projectname`

Por ejemplo, mi usuario es `crguezl`. Si el proyecto se llama `nathanuniversityexercisesPL`, la dirección será:

`http://crguezl.github.io/nathanuniversityexercisesPL/`

- Custom 404s will only work if a custom domain is used, otherwise the User Pages 404 is used.
- Creating Project Pages manually

1. Setting up Pages on a project requires a new `“orphan”` branch in your repository. The safest way to do this is to start with a fresh clone.

```

git clone https://github.com/user/repository.git
# Clone our repository
# Cloning into 'repository'...
remote: Counting objects: 2791, done.
remote: Compressing objects: 100% (1225/1225), done.
remote: Total 2791 (delta 1722), reused 2513 (delta 1493)
Receiving objects: 100% (2791/2791), 3.77 MiB | 969 KiB/s, done.
Resolving deltas: 100% (1722/1722), done.

```

2. Now that we have a clean repository, we need to create the new branch and remove all content from the working directory and index.

```
cd repository
```

```
git checkout --orphan gh-pages  
# Creates our branch, without any parents (it's an orphan!)  
# Switched to a new branch 'gh-pages'
```

```
git rm -rf .  
# Remove all files from the old working tree  
# rm '.gitignore'
```

3. Now we have an empty working directory. We can create some content in this branch and push it to GitHub. For example:

```
echo "My GitHub Page" > index.html  
git add index.html  
git commit -a -m "First pages commit"  
git push origin gh-pages
```

Capítulo 10

Functions and all that

<http://nathansuniversity.com/funcs.html>

Capítulo 11

Inventing a language for turtle graphics

<http://nathansuniversity.com/turtle.html>

Parte III

HERRAMIENTAS

Capítulo 12

Heroku

12.1. Introducción

Prerequisitos Estos son los prerequisites (Octubre 2013)

1. Basic Ruby knowledge, including an installed version of Ruby 2.0.0, Rubygems, and Bundler.
2. Basic Git knowledge
3. Your application must run on Ruby (MRI) 2.0.0.
4. Your application must use Bundler.
5. A Heroku user account.

Instala el Heroku Toolbelt

1. Crea una cuenta en Heroku
2. El Heroku Toolbelt se compone de:
 - a) Heroku client - CLI tool for creating and managing Heroku apps
 - b) Foreman - an easy option for running your apps locally
 - c) Git - revision control and pushing to Heroku

La primera vez te pedirá las credenciales:

```
$ heroku login
Enter your Heroku credentials.
Email: adam@example.com
Password:
Could not find an existing public key.
Would you like to generate one? [Yn]
Generating new SSH public key.
Uploading ssh public key /Users/adam/.ssh/id_rsa.pub
```

La clave la cargas en la sección SSH keys add key de <https://dashboard.heroku.com/account>

```
[~/rack/rack-rock-paper-scissors(test)]$ heroku --version
heroku-gem/2.39.4 (x86_64-darwin11.4.2) ruby/1.9.3
```

```
[~/local/src/ruby/sinatra/rack/rack-rock-paper-scissors(test)]$ which heroku
/Users/casiano/.rvm/gems/ruby-1.9.3-p392/bin/heroku
[~/local/src/ruby/sinatra/rack/rack-rock-paper-scissors(test)]$ ruby -v
ruby 1.9.3p392 (2013-02-22 revision 39386) [x86_64-darwin11.4.2]
```

Seguramente tienes que instalar una versión del toolbet por cada versión de Ruby con la que quieras usarlo.

Para desinstalarlo:

```
$ gem uninstall heroku --all
```

Actualizaciones The Heroku Toolbelt will automatically keep itself up to date.

1. When you run a heroku command, a background process will be spawned that checks a URL for the latest available version of the CLI.
2. If a new version is found, it will be downloaded and stored in `~/.heroku/client`.
3. This background check will happen at most once every 5 minutes.
4. The heroku binary will check for updated clients in `~/.heroku/client` before loading the system-installed version.

Ayuda

```
[~/local/src/ruby/sinatra/rack/rack-rock-paper-scissors(master)]$ heroku --help
Usage: heroku COMMAND [--app APP] [command-specific-options]
```

Primary help topics, type "heroku help TOPIC" for more details:

```
addons    # manage addon resources
apps      # manage apps (create, destroy)
auth      # authentication (login, logout)
config    # manage app config vars
domains   # manage custom domains
logs      # display logs for an app
ps        # manage dynos (dynos, workers)
releases  # manage app releases
run       # run one-off commands (console, rake)
sharing   # manage collaborators on an app
```

Additional topics:

```
account    # manage heroku account options
certs      # manage ssl endpoints for an app
db         # manage the database for an app
drains     # display syslog drains for an app
fork       # clone an existing app
git        # manage git for apps
help       # list commands and display help
keys       # manage authentication keys
labs       # manage optional features
maintenance # manage maintenance mode for an app
pg         # manage heroku-postgresql databases
pgbackups  # manage backups of heroku postgresql databases
plugins    # manage plugins to the heroku gem
regions    # list available regions
stack      # manage the stack for an app
status     # check status of heroku platform
update     # update the heroku client
version    # display version
```

Specify Ruby Version and Declare dependencies with a Gemfile

Heroku recognizes an app as Ruby by the existence of a **Gemfile**.

Even if your app has no gem dependencies, you should still create an empty **Gemfile** in order that it appear as a Ruby app.

In local testing, you should be sure to run your app in an isolated environment (via **bundle exec** or an empty RVM gemset), to make sure that all the gems your app depends on are in the **Gemfile**.

In addition to specifying dependencies, you'll want to specify your Ruby Version using the ruby DSL provided by Bundler.

Here's an example **Gemfile** for a Sinatra app:

```
source "https://rubygems.org"
ruby "2.0.0"
gem 'sinatra', '1.1.0'

[~/sinatra/rockpaperscissors(master)]$ cat Gemfile
source 'https://rubygems.org'
gem 'sinatra'
gem 'haml'
gem 'puma'
```

Run **bundle install** to set up your bundle locally.

1. Run:

```
$ bundle install
```

2. This ensures that all gems specified in **Gemfile**, together with their dependencies, are available for your application.
3. Running **bundle install** also generates a **Gemfile.lock** file, *which should be added to your git repository*.
4. **Gemfile.lock** ensures that your deployed versions of gems on Heroku match the version installed locally on your development machine.

Declare process types with Procfile

Process types are declared via a file named **Procfile** placed in the root of your app.

Its format is one process type per line, with each line containing:

<process type>: <command>

The syntax is defined as:

1. **<process type>** – an alphanumeric string, is a name for your command, such as
 - a) **web**,
 - b) **worker**,
 - c) **urgentworker**,
 - d) **clock**, etc.
2. **<command>** – a command line to launch the process, such as **rake jobs:work**.

The **web** process type is special as it's the only process type that will receive HTTP traffic from Heroku's routers.

1. Use a **Procfile**, a text file in the root directory of your application, to explicitly declare what command should be executed to start a web dyno.

2. Assume for instance, that we want to execute `web.rb` using Ruby. Here's a Procfile:

```
web: bundle exec ruby web.rb -p $PORT
```

3. If we are instead deploying a straight Rack app, here's a Procfile that can execute our `config.ru`:

```
web: bundle exec rackup config.ru -p $PORT

[~/sinatra/rockpaperscissors(spec)]$ cat config.ru
#\ -s puma
require './rps'
run RockPaperScissors::App
```

1. This declares a single process type, `web`, and the command needed to run it.
2. The name `web` is important here. It declares that this process type will be attached to the HTTP routing stack of Heroku, and receive web traffic when deployed.

Foreman

1. It's important when developing and debugging an application that the local development environment is executed in the same manner as the remote environments.
2. This ensures that incompatibilities and hard to find bugs are caught before deploying to production and treats the application as a holistic unit instead of a series of individual commands working independently.
3. Foreman is a command-line tool for running Procfile-backed apps. It's installed automatically by the Heroku Toolbelt.
4. If you had a Procfile with both `web` and `worker` process types, Foreman will start one of each process type, with the output interleaved on your terminal
5. We can now start our application locally using Foreman (installed as part of the Toolbelt):

```
$ foreman start
16:39:04 web.1      | started with pid 30728
18:49:43 web.1      | [2013-03-12 18:49:43] INFO  WEBrick 1.3.1
18:49:43 web.1      | [2013-03-12 18:49:43] INFO  ruby 2.0.0p247 (2013-06-27 revision 4167
18:49:43 web.1      | [2013-03-12 18:49:43] INFO  WEBrick::HTTPServer#start: pid=30728 por
```

6. Our app will come up on port 5000. Test that it's working with `curl` or a web browser, then `Ctrl-C` to exit.

Setting local environment variables

Config vars saved in the `.env` file of a project directory will be added to the environment when run by Foreman.

For example we can set the `RACK_ENV` to `development` in your environment.

```
$ echo "RACK_ENV=development" >>.env
$ foreman run irb
> puts ENV["RACK_ENV"]
> development
```

Do not commit the `.env` file to source control. It should only be used for local configuration.

Procfile y Despliegue

Véase la descripción de los contenidos del Procfile en 12.1.

1. A Procfile is not necessary to deploy apps written in most languages supported by Heroku.
2. The platform automatically detects the language, and creates a default web process type to boot the application server.
3. Creating an explicit Procfile is recommended for greater control and flexibility over your app.
4. For Heroku to use your Procfile, add the Procfile to the root of your application, then push to Heroku:

```
$ git add .
$ git commit -m "Procfile"
$ git push heroku
...
-----> Procfile declares process types: web, worker
        Compiled slug size is 10.4MB
-----> Launching... done
        http://strong-stone-297.herokuapp.com deployed to Heroku

To git@heroku.com:strong-stone-297.git
 * [new branch]      master -> master
```

Store your app in Git

```
$ git init
$ git add .
$ git commit -m "init"

[~/sinatra/rockpaperscissors(master)]$ git remote -v
origin  git@github.com:crguezl/sinatra-rock-paper-scissors.git (fetch)
origin  git@github.com:crguezl/sinatra-rock-paper-scissors.git (push)
```

Deploy your application to Heroku

Create the app on Heroku:

```
[~/sinatra/rockpaperscissors(master)]$ heroku create
Creating mysterious-falls-4594... done, stack is cedar
http://mysterious-falls-4594.herokuapp.com/ | git@heroku.com:mysterious-falls-4594.git
Git remote heroku added

[~/sinatra/rockpaperscissors(spec)]$ cat Rakefile
desc "start server using rackup ..."
task :default do
  sh "rackup"
end

require 'rspec/core/rake_task'

RSpec::Core::RakeTask.new do |task|
  task.rspec_opts = ["-c", "-f progress"]
  task.pattern     = 'spec/**/*.spec.rb'
end
```

```
[~/sinatra/rockpaperscissors(master)]$ git remote -v
heroku  git@heroku.com:mysterious-falls-4594.git (fetch)
heroku  git@heroku.com:mysterious-falls-4594.git (push)
origin  git@github.com:crguezl/sinatra-rock-paper-scissors.git (fetch)
origin  git@github.com:crguezl/sinatra-rock-paper-scissors.git (push)
```

Deploy your code:

```
[~/sinatra/rockpaperscissors(master)]$ git push heroku master
Counting objects: 31, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (29/29), done.
Writing objects: 100% (31/31), 9.09 KiB, done.
Total 31 (delta 11), reused 0 (delta 0)
```

```
-----> Ruby/Rack app detected
-----> Installing dependencies using Bundler version 1.3.2
Running: bundle install --without development:test --path vendor/bundle --binstubs vend
Fetching gem metadata from https://rubygems.org/.....
Fetching gem metadata from https://rubygems.org/..
Installing tilt (1.4.1)
Installing haml (4.0.3)
Installing rack (1.5.2)
Installing puma (2.0.1)
Installing rack-protection (1.5.0)
Installing sinatra (1.4.2)
Using bundler (1.3.2)
Your bundle is complete! It was installed into ./vendor/bundle
Post-install message from haml:
HEADS UP! Haml 4.0 has many improvements, but also has changes that may break
your application:
* Support for Ruby 1.8.6 dropped
* Support for Rails 2 dropped
* Sass filter now always outputs <style> tags
* Data attributes are now hyphenated, not underscored
* html2haml utility moved to the html2haml gem
* Textile and Maruku filters moved to the haml-contrib gem
For more info see:
http://rubydoc.info/github/haml/haml/file/CHANGELOG.md
Cleaning up the bundler cache.
-----> Discovering process types
Procfile declares types      -> (none)
Default types for Ruby/Rack -> console, rake, web

-----> Compiled slug size: 1.3MB
-----> Launching... done, v4
      http://mysterious-falls-4594.herokuapp.com deployed to Heroku
```

```
To git@heroku.com:mysterious-falls-4594.git
* [new branch]      master -> master
[~/sinatra/rockpaperscissors(master)]$
```

Visit your application

You've deployed your code to Heroku, and specified the process types in a Procfile.

You can now instruct Heroku to execute a process type.

Heroku does this by running the associated command in a dyno - a lightweight container which is the basic unit of composition on Heroku.

Let's ensure we have one dyno running the web process type:

```
$ heroku ps:scale web=1
```

Veamos que dice la ayuda:

```
$ heroku help ps
Usage: heroku ps
```

list processes for an app

Additional commands, type "heroku help COMMAND" for more details:

```
ps:restart [PROCESS]          # ps:restart [PROCESS]
ps:scale PROCESS1=AMOUNT1 ... # ps:scale PROCESS1=AMOUNT1 ...
ps:stop PROCESS               # ps:stop PROCESS
```

```
$ heroku help ps:scale
Usage: heroku ps:scale PROCESS1=AMOUNT1 ...
```

scale processes by the given amount

Example: `heroku ps:scale web=3 worker+1`

You can check the state of the app's dynos. The `heroku ps` command lists the running dynos of your application:

```
$ heroku ps
=== web: 'bundle exec ruby web.rb -p $PORT'
web.1: up for 9m
```

Here, one dyno is running.

```
[~/sinatra/sinatra-rock-paper-scissors/sinatra-rockpaperscissors(master)]$ heroku ps
Process  State          Command
-----  -
web.1    idle for 8h    bundle exec rackup config.ru -p $P..
```

We can now visit the app in our browser with `heroku open`.

```
[~/sinatra/rockpaperscissors(master)]$ heroku open
Opening http://mysterious-falls-4594.herokuapp.com/
[~/sinatra/rockpaperscissors(master)]$
```

Dyno sleeping and scaling

1. Having only a single web dyno running will result in the dyno going to sleep after one hour of inactivity.
2. This causes a delay of a few seconds for the first request upon waking.
3. Subsequent requests will perform normally.
4. To avoid this, you can scale to more than one web dyno. For example:

```
$ heroku ps:scale web=2
```

5. For each application, Heroku provides 750 free dyno-hours.
6. Running your app at 2 dynos would exceed this free, monthly allowance, so let's scale back:

```
$ heroku ps:scale web=1
```

View the logs

Heroku treats logs as streams of time-ordered events aggregated from the output streams of all the dynos running the components of your application.

Heroku's Logplex provides a single channel for all of these events.

View information about your running app using one of the logging commands, `heroku logs`:

```
$ heroku logs
```

```
2013-03-13T04:10:49+00:00 heroku[web.1]: Starting process with command 'bundle exec ruby web.r
2013-03-13T04:10:50+00:00 app[web.1]: [2013-03-13 04:10:50] INFO WEBrick 1.3.1
2013-03-13T04:10:50+00:00 app[web.1]: [2013-03-13 04:10:50] INFO ruby 2.0.0p247 (2013-06-27 r
2013-03-13T04:10:50+00:00 app[web.1]: [2013-03-13 04:10:50] INFO WEBrick::HTTPServer#start: p
```

Console

1. Heroku allows you to run commands in a one-off dyno - scripts and applications that only need to be executed when needed - using the `heroku run` command.
2. You can use this to launch an interactive Ruby shell (`bundle exec irb`) attached to your local terminal for experimenting in your app's environment:

```
$ heroku run console
Running 'console' attached to terminal... up, ps.1
irb(main):001:0>
```

3. By default, `irb` has nothing loaded other than the Ruby standard library. From here you can require some of your application files. Or you can do it on the command line:

```
$ heroku run console -r ./web
```

Rake

Rake can be run in an attached dyno exactly like the console:

```
$ heroku run rake db:migrate
```

Using a SQL database

By default, non-Rails apps aren't given a SQL database.

This is because you might want to use a NoSQL database like Redis or CouchDB, or you don't need any database at all.

If you need a SQL database for your app, do this:

1. `$ heroku addons:add heroku-postgresql:dev`
2. You must also add the Postgres gem to your app in order to use your database. Add a line to your `Gemfile` like this:

```
gem 'pg'
```

3. You'll also want to setup a local PostgreSQL database.

Webserver

By default your app (Rack) will use **Webrick**.

This is fine for testing, but for production apps you'll want to switch to a more robust webserver. On Cedar, they recommend **Unicorn** as the webserver.

12.2. Logging

Heroku aggregates three categories of logs for your app:

1. App logs - Output from your application.

This will include logs generated from

- a)* within your application,
- b)* application server and
- c)* libraries.

(Filter: `--source app`)

2. System logs -

Messages about actions taken by the Heroku platform infrastructure on behalf of your app, such as:

- a)* restarting a crashed process,
- b)* sleeping or waking a web dyno, or
- c)* serving an error page due to a problem in your app.

(Filter: `--source heroku`)

3. API logs -

Messages about administrative actions taken by you and other developers working on your app, such as:

- a)* deploying new code,
- b)* scaling the process formation, or
- c)* toggling maintenance mode.

(Filter: `--source heroku --ps api`)

```
[~/rack/rack-rock-paper-scissors(master)]$ heroku logs --source heroku --ps api
2013-10-23T21:33:41.105090+00:00 heroku[api]: Deploy 5ec1351 by chuchu.chachi.leon@gmail.
2013-10-23T21:33:41.154690+00:00 heroku[api]: Release v7 created by chuchu.chachi.leon@gmail.
```

Logplex is designed for collating and routing log messages, not for storage. It keeps the last 1,500 lines of consolidated logs.

Heroku recommends using a separate service for long-term log storage; see Syslog drains for more information.

Writing to your log

Anything written to standard out (stdout) or standard error (stderr) is captured into your logs. This means that you can log from anywhere in your application code with a simple output statement:

```
puts "Hello, logs!"
```

To take advantage of the realtime logging, you may need to disable any log buffering your application may be carrying out. For example, in Ruby add this to your config.ru:

```
$stdout.sync = true
```

Some frameworks send log output somewhere other than stdout by default.

To fetch your logs

```
$ heroku logs
2010-09-16T15:13:46.677020+00:00 app[web.1]: Processing PostController#list (for 208.39.138.12
2010-09-16T15:13:46.677023+00:00 app[web.1]: Rendering template within layouts/application
2010-09-16T15:13:46.677902+00:00 app[web.1]: Rendering post/list
2010-09-16T15:13:46.678990+00:00 app[web.1]: Rendered includes/_header (0.1ms)
2010-09-16T15:13:46.698234+00:00 app[web.1]: Completed in 74ms (View: 31, DB: 40) | 200 OK [ht
2010-09-16T15:13:46.723498+00:00 heroku[router]: at=info method=GET path=/posts host=myapp.her
2010-09-16T15:13:47.893472+00:00 app[worker.1]: 2 jobs processed at 16.6761 j/s, 0 failed ...
```

In this example, the output includes log lines from one of the app's web dynos, the Heroku HTTP router, and one of the app's workers.

The logs command retrieves 100 log lines by default.

Log message ordering

When retrieving logs, you may notice that the logs are not always in order, especially when multiple components are involved.

This is likely an artifact of distributed computing.

Logs originate from many sources (router nodes, dynos, etc) and are assembled into a single log stream by logplex.

It is up to the logplex user to sort the logs and provide the ordering required by their application, if any

Log history limits

You can fetch up to 1500 lines using the `-num` (or `-n`) option:

```
$ heroku logs -n 200
```

Heroku only stores the last 1500 lines of log history. If you'd like to persist more than 1500 lines, use a logging add-on or create your own syslog drain¹.

Log format

Each line is formatted as follows:

1. timestamp source[dyno]: message
2. Timestamp - The date and time recorded at the time the log line was produced by the dyno or component. The timestamp is in the format specified by RFC5424, and includes microsecond precision.
3. Source -
 - a) All of your app's dynos (web dynos, background workers, cron) have a source of app.
 - b) All of Heroku's system components (HTTP router, dyno manager) have a source of heroku.
4. Dyno - The name of the dyno or component that wrote this log line. For example, **worker #3** appears as **worker.3**, and the Heroku HTTP router appears as router.
5. Message - The content of the log line. Dynos can generate messages up to approximately 1024 bytes in length and longer messages will be truncated.

¹Logplex drains allow you to forward your Heroku logs to an external syslog server for long-term archiving. You must configure the service or your server to be able to receive syslog packets from Heroku, and then add its syslog URL (which contains the host and port) as a syslog drain.

Realtime tail

1. Similar to `tail -f`, realtime tail displays recent logs and leaves the session open for realtime logs to stream in.
2. By viewing a live stream of logs from your app, you can gain insight into the behavior of your live application and debug current problems.
3. You may tail your logs using `--tail` (or `-t`).

```
$ heroku logs --tail
```

When you are done, press Ctrl-C to close the session.

Filtering

If you only want to fetch logs with a certain source, a certain dyno, or both, you can use the `--source` (or `-s`) and `--ps` (or `-p`) filtering arguments:

```
$ heroku logs --ps router
```

```
2012-02-07T09:43:06.123456+00:00 heroku[router]: at=info method=GET path=/stylesheets/dev-cent
2012-02-07T09:43:06.123456+00:00 heroku[router]: at=info method=GET path=/articles/bundler hos
```

```
$ heroku logs --source app
```

```
2012-02-07T09:45:47.123456+00:00 app[web.1]: Rendered shared/_search.html.erb (1.0ms)
2012-02-07T09:45:47.123456+00:00 app[web.1]: Completed 200 OK in 83ms (Views: 48.7ms | ActiveR
2012-02-07T09:45:47.123456+00:00 app[worker.1]: [Worker(host:465cf64e-61c8-46d3-b480-362bfd4ec
2012-02-07T09:46:01.123456+00:00 app[web.6]: Started GET "/articles/buildpacks" for 4.1.81.209
```

```
$ heroku logs --source app --ps worker
```

```
2012-02-07T09:47:59.123456+00:00 app[worker.1]: [Worker(host:260cf64e-61c8-46d3-b480-362bfd4ec
2012-02-07T09:47:59.123456+00:00 app[worker.1]: [Worker(host:260cf64e-61c8-46d3-b480-362bfd4ec
```

When filtering by dyno, either the base name, `--ps web`, or the full name, `--ps web.1`, may be used.

You can also combine the filtering switches with `--tail` to get a realtime stream of filtered output.

```
$ heroku logs --source app --tail
```

12.3. Troubleshooting

If you push your app and it crashes, `heroku ps` shows state crashed:

```
=== web (1X): 'bundle exec thin start -R config.ru -e $RACK_ENV -p $PORT'
web.1: crashed 2013/10/24 20:21:34 (~ 1h ago)
```

check your logs to find out what went wrong.

Here are some common problems.

Failed to require a sourcefile

If your app failed to require a sourcefile, chances are good you're running Ruby 1.9.1 or 1.8 in your local environment.

The load paths have changed in Ruby 1.9 which applies to Ruby 2.0.

Port your app forward to Ruby 2.0.0 making certain it works locally before trying to push to Cedar again.

Encoding error Ruby 1.9 added more sophisticated encoding support to the language which applies to Ruby 2.0.

Not all gems work with Ruby 2.0. If you hit an encoding error, you probably haven't fully tested your app with Ruby 2.0.0 in your local environment.

Port your app forward to Ruby 2.0.0 making certain it works locally before trying to push to Cedar again.

Missing a gem

If your app crashes due to missing a gem, you may have it installed locally but not specified in your Gemfile.

You must isolate all local testing using `bundle exec`.

For example, don't run `ruby web.rb`, run

```
bundle exec ruby web.rb
```

Don't run `rake db:migrate`, run

```
bundle exec rake db:migrate.
```

Another approach is to create a blank RVM gemset to be absolutely sure you're not touching any system-installed gems:

```
$ rvm gemset create myapp
```

```
$ rvm gemset use myapp
```

Runtime dependencies on development/test gems

If you're still missing a gem when you deploy, check your Bundler groups.

Heroku builds your app without the `development` or `test` groups, and if your app depends on a gem from one of these groups to run, you should move it out of the group.

One common example using the `RSpec` tasks in your Rakefile. If you see this in your Heroku deploy:

```
$ heroku run rake -T
Running 'rake -T' attached to terminal... up, ps.3
rake aborted!
no such file to load -- rspec/core/rake_task
```

Then you've hit this problem.

First, duplicate the problem locally like so:

```
$ bundle install --without development:test
...
$ bundle exec rake -T
rake aborted!
no such file to load -- rspec/core/rake_task
```

Now you can fix it by making these Rake tasks conditional on the gem load. For example:

```
begin
  require "rspec/core/rake_task"

  desc "Run all examples"
  RSpec::Core::RakeTask.new(:spec) do |t|
    t.rspec_opts = %w[--color]
    t.pattern = 'spec/*_spec.rb'
  end
rescue LoadError
end
```

Confirm it works locally, then push to Heroku.

Rack::Sendfile

Heroku does not support the use of Rack::Sendfile.

Rack::Sendfile usually requires that there is a frontend webserver like nginx or apache is running on the same machine as the application server.

This is not how Heroku is architected. Using the Rack::Sendfile middleware will cause your file downloads to fail since it will send a body with Content-Length of 0.

12.4. Configuration

```
[~/sinatra/sinatra-datamapper-jump-start(master)]$ heroku help config
Usage: heroku config
```

display the config vars for an app

-s, --shell # output config vars in shell format

Examples:

```
$ heroku config
```

```
A: one
```

```
B: two
```

```
$ heroku config --shell
```

```
A=one
```

```
B=two
```

Additional commands, type "heroku help COMMAND" for more details:

```
config:get KEY # display a config value for an app
config:set KEY1=VALUE1 [KEY2=VALUE2 ...] # set one or more config vars
config:unset KEY1 [KEY2 ...] # unset one or more config vars
```

```
[~/sinatra/sinatra-datamapper-jump-start(master)]$ heroku config -s
DATABASE_URL=postgres://bhhatrhjjhwcvt:hjgjfhhgjfhjfuWH7ls_PJKK5QD@ec2-54-204-35-132.compute-1.
HEROKU_POSTGRESQL_BLACK_URL=postgres://bhjshfdhakwcvt:hQssnhq1y1jhgfhgls_PGNu5QD@ec2-54-204-35
```

```
[~/sinatra/sinatra-datamapper-jump-start(master)]$ heroku config:set C=4
Setting config vars and restarting crguezl-songs... done, v6
```

```
C: 4
```

```
[~/sinatra/sinatra-datamapper-jump-start(master)]$ heroku config:get C
4
```

```
[~/sinatra/sinatra-datamapper-jump-start(master)]$ heroku config:unset C
Unsetting C and restarting crguezl-songs... done, v7
```

```
[~/sinatra/sinatra-datamapper-jump-start(master)]$ heroku config:get C
```

```
[~/sinatra/sinatra-datamapper-jump-start(master)]$]]
```

12.5. Make Heroku run non-master Git branch

Make Heroku run non-master Git branch You can push an alternative branch to Heroku using Git.

```
git push heroku-dev test:master
```

This pushes your local test branch to the remote's master branch (on Heroku).

El manual de `git push` dice:

To push a local branch to an established remote, you need to issue the command:

```
git push <REMOTENAME> <BRANCHNAME>
```

This is most typically invoked as `git push origin master`.

If you would like to give the branch a different name on the upstream side of the push, you can issue the command:

```
git push <REMOTENAME> <LOCALBRANCHNAME>:<REMOTEBRANCHNAME>
```

12.6. Account Verification and add-ons

You must verify your account by adding a credit card before you can add any add-on to your app other than `heroku-postgresql:dev` and `pgbackups:plus`.

Adding a credit card to your account lets you

1. use the free add-ons,
2. allows your account to have more than 5 apps at a time (verified accounts may have up to 100 apps),
3. and gives you access to turn on paid services any time with a few easy clicks.
4. The easiest way to do this is to go to your account page and click `Add Credit Card`.
5. Alternatively, when you attempt to perform an action that requires a credit card, either from the Heroku CLI or through the web interface, you will be prompted to visit the credit card page.

```
[~/sinatra/sinatra-datamapper-jump-start(master)]$ heroku addons:add rediscloud:20
Adding rediscloud:20 on dgjgxc1-songs... failed
!   Please verify your account to install this add-on
!   For more information, see http://devcenter.heroku.com/categories/billing
!   Verify now at https://heroku.com/verify
```

12.7. Véase

- Heroku: Getting Started with Ruby on Heroku
- SitePoint: Get Started with Sinatra on Heroku by Jagadish Thaker. Published August 12, 2013
- Deploying Rack-based Apps
- Heroku: List of Published Articles for Ruby
- Foreman
 1. Introducing Foreman by David Dollar
 2. Foreman man pages
 3. Applying the Unix Process Model to Web Apps by Adam Wiggins
- Ruby Kickstart - Session 6 de Joshua Cheek (Vimeo)
- sinatra-rock-paper-scissors
- The Procfile is your friend 13 January, 2012. Neil Middleton

Parte IV

PARTE: BITÁCORA DEL CURSO

Capítulo 13

2014

13.1. 01

13.1.1. Semana del 27/01/14 al 01/02/2014

- Presentación de la Asignatura
- Ejercicio: Darse de alta en la comunidad de google plus PL Grado ULL 13/14
- JavaScript Review
- *Expresiones Regulares y Análisis Léxico en JavaScript* 1
- *Conversor de Temperaturas* 1.2
- *GitHub Project Pages* 9.4.4

13.2. 02

13.2.1. Semana del 4/02/14 al 7/02/2014

- Martes 4/02. *Comma Separated Values. CSV* Sección 1.3. Secciones: Donde, Introducción al formato CSV, Ejemplo de ejecución, Aproximación al análisis mediante expresiones regulares de CSV.

13.2.2. Semana del 24/02/14 al 02/03/14. Repaso para el micro-examen del 05/03/14

1. ¿Que retorna?

```
"hello small world and blue sky".match(/(\S+)\s+(\S+)/);
```

2. Indique que casa con el primer paréntesis y que con el segundo en las siguientes expresiones regulares:

```
> x = "I have 2 numbers: 53147"
> pats = [ /(.*)(\d*)/,
           /(.*)(\d+)/,
           /(.*?)(\d*)/,
           /(.*?)(\d+)/,
           /(.*)(\d+)$/,
           /(.*?)(\d+)$/,
           /(.*)\b(\d+)$/,
           /(.*\D)(\d+)$/ ]
```

Es decir, compute la salida de:

```
pats.map( function(r) { return r.exec(x).slice(1); })
```

3. ¿Que retorna el matching?:

```
> a = "hola juan"
=> "hola juan"
> a.match(/(?:hola )*(juan)/)
```

4. ¿Que salidas se obtienen?

```
> "a\na".match(/a$/ )
-----
> "a\na".match(/a$/m)
-----
> "a\na".match(/^a/gm)
-----
> "a\na".match(/^a/g)
-----
```

5. Escriba la expresión regular que da lugar a este resultado (enumerar las líneas):

```
> x = "one\ntwo\nthree\nfour"
'one\ntwo\nthree\nfour'
> a = (c = 1, x.replace(_____, function(t) { return  c++ + ' ' + t; }))
'1 one\n2 two\n3 three\n4 four'
> console.log(a)
1 one
2 two
3 three
4 four
undefined
```

6. Supongamos dado el método

```
String.prototype.repeat = function( num ) {
  return new Array( num + 1 ).join( this );
}
```

de manera que podamos escribir expresiones como:

```
> x = 'a'.repeat(40)
'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa'
```

Encontremos una solución de la ecuación diofántica $3x + 2y + 5z = 40$

```
> m = x.match(/^_____$/).slice(1)
[ 'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa',
  'aa',
  'aaaaa' ]
```

Calculemos las longitudes de las tres cadenas:

```
> r = m.map(function(s) { return s.length; })
[ 33, 2, 5 ]
```

Dividamos por los coeficientes para obtener la solución:

```
> coef = [3, 2, 5]
> i = 0; w = r.map(function(x) { return x/coef[i++]; })
[ 11, 1, 1 ]
```

Encuentre la expresión regular usada.

7. Escriba una expresión regular que reconozca cadenas de dobles comillas como "hello world" y en las que las comillas puedan aparecer escapadas como en "Hello \"Jane\" and Jakes"
8. Escriba una expresión regular que reconozca los números en punto flotante como 2.34, -5.2e-1 y 0.9e3
9. ¿Que queda en m[0]?

```
m = 'main() /* 1c */ { /* 2c */ return; /* 3c */ }'.match(new RegExp('/\\\*..*\\\*/'))
```

¿Por qué?

10. ¿Por qué debemos duplicar el carácter de escape \ en la expresión regular new RegExp('/*..**/') de la pregunta anterior 9?
11. Se quiere poner un espacio en blanco después de la aparición de cada coma:

```
> 'ab,cd,4,3, de, fg'.replace(/,/ , ' ', ' ')
=> "ab, cd, 4, 3, de, fg"
```

pero se quiere que la sustitución no tenga lugar si la coma esta incrustada entre dos dígitos. Además se pide que si hay ya un espacio después de la coma, no se duplique

Como función de reemplazo use:

```
f = function(match, p1, p2, offset, string) { return (p1 || p2 + " "); }
```

12. Escribe un patrón regular que reconozca las cadenas que representan números no primos en unario de manera que el primer paréntesis case con el divisor mas grande del número.
13. Escribe un patrón regular que reconozca las cadenas que representan números no primos en unario de manera que el primer paréntesis case con el divisor mas pequeño del número.
14. Escriba una expresión regular que reconozca los comentarios del lenguaje JavaScript de la forma // ...
15. Escriba una expresión regular que reconozca los comentarios del lenguaje JavaScript de la forma /* ... */
16. Rellene lo que falta para que la salida sea la que aparece en la sesión de node:

```
> re = _____
> str = "John Smith"
'John Smith'
> newstr = str.replace(re, "_____")
'Smith, John'
```

17. Rellene las partes que faltan:

```
> re = /d(b+)(d)/ig
/d(b+)(d)/gi
> z = "dBdxdbbdzdbd"
'dBdxdbbdzdbd'
> result = re.exec(z)
[ _____, _____, _____, index: __, input: 'dBdxdbbdzdbd' ]
> re.lastIndex
_____
> result = re.exec(z)
[ _____, _____, _____, index: __, input: 'dBdxdbbdzdbd' ]
> re.lastIndex
_____
> result = re.exec(z)
[ _____, _____, _____, index: __, input: 'dBdxdbbdzdbd' ]
> re.lastIndex
_____
> result = re.exec(z)
_____
```

18. Escriba la expresión regular `r` para que produzca el resultado final:

```
> x = "hello"
> r = /l(____)/
> z = r.exec(x)
[ 'l', index: 3, input: 'hello' ]
```

19. > z = "dBdDBBD"

```
> re = /d(b+)(d)/ig
> re.lastIndex = _____
> result = re.exec(z)
[ 'DBBD',
  'BB',
  'D',
  index: 3,
  input: 'dBdDBBD' ]
```

20. Conteste:

a) Explique que hace el siguiente fragmento de código:

```
> RegExp.prototype.bexec = function(str) {
...   var i = this.lastIndex;
...   var m = this.exec(str);
...   if (m && m.index == i) return m;
...   return null;
... }
[Function]
```

b) Rellene las salidas que faltan:

```
> re = /d(b+)(d)/ig
/d(b+)(d)/gi
> z = "dBdXXXXDBBD"
'dBdXXXXDBBD'
```



```
> re.lastIndex = 3
> re.bexec(z)
```

```
-----
> re.lastIndex = 7
> re.bexec(z)
-----
```

21. Escriba una expresión JavaScript que permita reemplazar todas las apariciones de palabras repetidas en una String por una sólo aparición de la misma
22. Supongamos que se usa una función como segundo argumento de `replace`. ¿Que argumentos recibe?
23. ¿Cual es la salida?

```
> "bb".match(/b|bb/)
> "bb".match(/bb|b/)
```

24. El siguiente fragmento de código tiene por objetivo escapar las entidades HTML para que no sean interpretadas como código HTML. Rellene las partes que faltan.

```
var entityMap = {
  "&": "&__;",
  "<": "&__;",
  ">": "&__;",
  "'": '"',
  '"': '"',
  "/": '"'
};

function escapeHtml(string) {
  return String(string).replace(/_____/g, function (s) {
    return _____;
  });
}
```

25. ¿Cual es la salida?

```
> a = [1,2,3]
[ 1, 2, 3 ]
> b = [1,2,3]
[ 1, 2, 3 ]
> a == b
-----
```

26. ¿Como se llama el método que permite obtener una representación como cadena de un objeto? ¿Que parámetros espera? ¿Como afectan dichos parámetros?
27. ¿Cual debe ser el valor del atributo `rel` para usar la imagen como favicon?

```
<link rel="_____" href="etsiull.png" type="image/x-icon">
```

28. Escriba un código JavaScript que defina una clase `Persona` con atributos `nombre` y `apellidos` y que disponga de un método `saluda`.

29. Reescriba la solución al problema anterior haciendo uso del método `template` de `underscore` y ubicando el template dentro de un tag `script`.
30. Rellene lo que falta:

```
[~/srcPLgrado/temperature/tests(master)]$ cat tests.js
var assert = chai._____;

suite('temperature', function() {
  test('[1,{a:2}] == [1,2]', function() {
    assert._____( [1, {a:2}], [1, {a:2}]);
  });
  test('5X = error', function() {
    original.value = "5X";
    calculate();
    assert._____(converted.innerHTML, /ERROR/);
  });
});
```

31. ¿Cómo se llama el directorio por defecto desde el que una aplicación sinatra sirve los ficheros estáticos?
32. Explique la línea:

```
set :public_folder, File.dirname(__FILE__) + '/starterkit'
```

¿Que es `__FILE__`? ¿Que es `File.dirname(__FILE__)`? ¿Que hace el método `set`? (Véase <http://www.sinatrarb.com/configuration.html>)

33. Escriba un programa sinatra que cuando se visite la URI `/chuchu` muestre una página que diga `"hello world!"`
34. ¿Cual es el significado de `__END__` en un programa Ruby?
35. Esta y las preguntas 36 y 37 se refieren al mismo programa ruby sinatra. Explique este fragmento de dicho programa ruby sinatra.

```
@@layout
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>Demo</title>
  </head>
  <body>
    <a href="http://jquery.com/">jQuery</a>
    <div class="result"></div>
    <script src="jquery.js"></script>
    <%= yield %>
  </body>
</html>
```

- a) ¿En que lugar del fichero que contiene el programa está ubicada esta sección?
- b) ¿Cómo se llama el lenguaje en el que esta escrita esta sección?
- c) ¿Para que sirve la sección `layout`?

- d) ¿Cual es la función del `<div class="result"></div>`?
- e) ¿Para que sirve el `<%= yield %>`?

36. Explique este fragmento de un programa ruby sinatra.

```
@@index
<script>
$( document ).ready(function() {
    $( "a" ).click(function( event ) {
        event.preventDefault();
        $.get( "/chuchu", function( data ) {
            $( ".result" ).html( data );
            alert( "Load was performed." );
        });
    });
});
</script>
```

- a) ¿Cuando ocurre el evento `ready`?
- b) ¿Que hace `event.preventDefault()`?
- c) ¿Que hace `$.get("/chuchu", function(data) { ... })`? ¿Cuando se dispara la callback?
- d) ¿que hace la línea `$(".result").html(data)`?

37. Explique este fragmento de código ruby-sinatra:

```
get '/chuchu' do
  if request.xhr?
    "hello world!"
  else
    erb :tutu
  end
end
```

38. En el siguiente programa - que calcula la conversión de temperaturas entre grados Farenheit y Celsius - rellene las partes que faltan:

a) index.html:

```
<html>
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=____">
  <title>JavaScript Temperature Converter</title>
  <link ____="global.css" ____="stylesheet" ____="text/css">

  <script type="____" src="temperature.js"></script>
</head>
<____>
  <h1>Temperature Converter</h1>
  <table>
    <tr>
      <th>Enter Temperature (examples: 32F, 45C, -2.5f):</th>
      <td><input id="____" ____="calculate();"></td>
    </tr>
```

```

        <tr>
            <th>Converted Temperature:</th>
            <td><span class="output" id="_____"></span></td>
        </tr>
    </table>
</____>
</html>

```

b) Rellene las partes del código JavaScript que faltan en `temperature.js`:

```

"use strict"; // Use ECMAScript 5 strict mode in browsers that support it
function calculate() {
    var result;
    var original = document.getElementById("_____");
    var temp = original.value;
    var regexp = /_____/;

    var m = temp.match(_____);

    if (m) {
        var num = ____; // paréntesis correspondiente
        var type = ____;
        num = parseFloat(num);
        if (type == 'c' || type == 'C') {
            result = (num * 9/5)+32;
            result = _____ // 1 sólo decimal y el tipo
        }
        else {
            result = (num - 32)*5/9;
            result = _____ // 1 sólo decimal y el tipo
        }
        converted._____ = result; // Insertar "result" en la página
    }
    else {
        converted._____ = "ERROR! Try something like '-4.2C' instead";
    }
}

```

39. ¿Que hace autofocus?

```

<td><textarea autofocus cols = "80" rows = "5" id="original"></textarea></td>

```

40. ¿Que hacen las siguientes pseudo-clases estructurales CSS3?

```

tr:nth-child(odd)    { background-color:#eee; }
tr:nth-child(even)   { background-color:#00FF66; }

```

41. ¿Que contiene el objeto `window` en un programa JavaScript que se ejecuta en un navegador?

42. a) ¿Que es Local Storage? ¿Que hace la siguiente línea?

```

if (window.localStorage) localStorage.original = temp;

```

b) ¿Cuándo se ejecutará esta callback? ¿Que hace?

```

window.onload = function() {
    // If the browser supports localStorage and we have some stored data

```

```

        if (window.localStorage && localStorage.original) {
            document.getElementById("original").value = localStorage.original;
        }
    };

```

43. ¿Cómo se hace para que elementos de la página web permanezcan ocultos para posteriormente mostrarlos? ¿Que hay que hacer en el HTML, en la hoja de estilo y en el JavaScript?

44. Rellene los estilos para los elementos de las clases para que su visibilidad case con la que su nombre indica:

```

.hidden      { display: ____; }
.unhidden   { display: _____; }

```

45. Los siguientes textos corresponden a los ficheros de la práctica de construcción de un analizador léxico de los ficheros de configuración INI. Rellena las partes que faltan.

a) Rellena las partes que faltan en el contenido del fichero `index.html`. Comenta que hace el tag `<input>`. Comenta que hace el tag `<pre>`.

```

<html>
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>INI files</title>
    <link href="global.css" rel="_____" type="text/css">

    <script type="_____" src="underscore.js"></script>
    <script type="_____" src="jquery.js"></script>
    <script type="_____" src="_____"></script>
</head>
<body>
    <h1>INI files</h1>
    <input type="file" id="_____" />
    <div id="out" class="hidden">
        <table>
            <tr><th>Original</th><th>Tokens</th></tr>
            <tr>
                <td>
                    <pre class="input" id="_____"></pre>
                </td>
                <td>
                    <pre class="output" id="_____"></pre>
                </td>
            </tr>
        </table>
    </div>
</body>
</html>

```

b) A continuación siguen los contenidos del fichero `ini.js` conteniendo el JavaScript.

1) Rellena las partes que faltan. El siguiente ejemplo de fichero `.ini` le puede ayudar a recordar la parte de las expresiones regulares

```

; last modified 1 April 2001 by John Doe
[owner]
name=John Doe
organization=Acme Widgets Inc.

```

- 2) Explica el uso del template.
- 3) Explica el uso de JSON.stringify

```
"use _____"; // Use ECMAScript 5 strict mode in browsers that support it

$(document)._____(function() {
    $("#fileinput")._____(calculate);
});

function calculate(evt) {
    var f = evt.target.files[0];

    if (f) {
        var r = new _____();
        r.onload = function(e) {
            var contents = e.target._____;

            var tokens = lexer(contents);
            var pretty = tokensToString(tokens);

            out.className = 'unhidden';
            initialinput._____ = contents;
            finaloutput._____ = pretty;
        }
        r._____(f); // Leer como texto
    } else {
        alert("Failed to load file");
    }
}

var temp = '<li> <span class = "<%= _____ %>"> <%= _ %> </span>\n';

function tokensToString(tokens) {
    var r = '';
    for(var i in tokens) {
        var t = tokens[i];
        var s = JSON.stringify(t, undefined, 2); //_____
        s = _.template(temp, {t: t, s: s});
        r += s;
    }
    return '<ol>\n'+r+'</ol>';
}

function lexer(input) {
    var blanks      = /^___/;
    var iniheader   = /^_____/;
    var comments    = /^_____/;
    var nameEqualValue = /^_____/;
    var any         = /^_____/;

    var out = [];
    var m = null;

    while (input != '') {
```

```

if (m = blanks.____(input)) {
    input = input.substr(m.index+_____);
    out.push({ type : _____, match: _ });
}
else if (m = iniheader.exec(input)) {
    input = input.substr(_____);
    _____ // avanzemos en input
}
else if (m = comments.exec(input)) {
    input = input.substr(_____);
    _____
}
else if (m = nameEqualValue.exec(input)) {
    input = input.substr(_____);
    _____
}
else if (m = any.exec(input)) {
    _____
    input = '';
}
else {
    alert("Fatal Error!" + substr(input,0,20));
    input = '';
}
}
return out;
}

```

13.3. Proyecto: Diseña e Implementa un Lenguaje de Dominio Específico

Se trata de realizar un proyecto relacionado con el procesamiento de lenguajes. El objetivo puede ser:

1. Diseñar un lenguaje de dominio específico para simplificar cualquier tarea en la que estés interesado:
 - Para escribir exámenes,
 - Por ejemplo se puede escribir un traductor para el formato Moodle gift que traduzca a javascript + HTML + css y que evalúe al usuario
 - Por ejemplo se puede escribir un traductor para el formato Moodle XML que traduzca a javascript + HTML + css y que evalúe al usuario
 -
 - Para dibujar árboles,
 - Para calcular fechas,
 - Para generar emails
 - Para escribir música
 - Para escribir autómatas finitos
 - Para procesar CSS
 - etc.

2. Estudiar un traductor existente en profundidad como:

- ECMAScript 5.1: Creating a JavaScript Parser Una implementación de ECMAScript 5.1 usando Jison disponible en GitHub en <https://github.com/cjihrig/jsparser>. Puede probarse en: <http://www.cjihrig.com/development/jsparser/>
- Roy
- CoffeeScript
- Jison
- Javascript 1.4
- etc.

3. También puedes proponer tu propio tema relacionado al profesor

Se recomienda para ello organizar equipos de no menos de dos y no mas de cuatro.

Las presentaciones de los proyectos tendrán lugar el último día de clase Martes 21 de Mayo.

Índice general

Índice de figuras

Índice de cuadros

Índice alfabético

- árbol sintáctico concreto, 52, 85
- access link, 108
- acción de reducción, 99
- acciones de desplazamiento, 99
- acciones semánticas, 66
- acciones shift, 99
- algoritmo de construcción del subconjunto, 98
- antiderivación, 94
- atributo heredado, 67, 128, 131
- atributo sintetizado, 66, 128, 131
- atributos formales, 131
- atributos heredados, 128, 129, 131
- atributos intrínsecos, 131
- atributos sintetizados, 128, 131
- autómata finito determinista, 98
- autómata finito no determinista con ϵ -transiciones, 96
- bubble phase, 43
- callback, 20
- capture phase, 43
- clausura, 98
- conflicto de desplazamiento-reducción, 100, 122
- conflicto reduce-reduce, 100, 122
- conflicto shift-reduce, 100, 122
- definición dirigida por la sintaxis, 131
- devDependencies, 155
- DFA, 98
- documento aqui, 65
- DOM storage, 29
- Ejercicio
 - Ambigüedad y LL(1), 60
 - Calcular los *FOLLOW*, 58
 - Caracterización de una gramática LL(1), 59
 - Construir los *FIRST*, 58
 - Factores Comunes, 56
 - Recorrido del árbol en un ADPR, 56
- esquema de traducción, 66, 127
- evaluation stack, 108
- favicon, 33
- Favorite icon, 33
- función de transición del autómata, 98
- goto, 99
- grafo de dependencias, 132
- gramática atribuida, 132
- gramática es recursiva por la izquierda, 67, 71
- handle, 95
- here document, 65
- INI, 40
- items núcleo, 103
- JavaScript Object Notation, 25
- jQuery, 18
- JSON, 25
- Karma, 149
- L-atribuida, 132
- LALR, 101
- lenguaje de las formas sentenciales a rderechas, 95
- LL(1), 59
- local storage, 29
- LR, 94
- máximo factor común, 56
- manecilla, 95
- mango, 95
- Mocha TDD interface, 149
- NFA, 96
- orden parcial, 132
- orden topológico, 132
- parsing expression, 67
- parsing expression grammar, 67
- pattern matching, 31
- PEG, 67
- Práctica
 - Ambigüedad en C++, 82
 - Analizador Léxico para Un Subconjunto de JavaScript, 49
 - Calculadora con Análisis de Ámbito, 106
 - Calculadora con Funciones, 105
 - Calculadora con Listas de Expresiones y Variables, 94
 - Comma Separated Values. CSV, 13

- Construcción de los FIRST y los FOLLOW, 58
- Conversor de Temperaturas, 9
- Ficheros INI, 41
- Generación Automática de Analizadores Predictivos, 60
- Inventando un Lenguaje: Tortoise, 84
- Palabras Repetidas, 36
- Secuencia de Asignaciones Simples, 89
- Traducción de Infijo a Postfijo, 104
- Un analizador APDR, 60
- Primeros, 97
- recursion, 108
- recursiva por la derecha, 77
- recursiva por la izquierda, 67, 71
- recursive descent parser, 68
- reducción por defecto, 114
- reducción-reducción, 100, 122
- reentrant, 108
- register spilling, 108
- reglas de evaluación de los atributos, 131
- reglas semánticas, 131
- rightmost derivation, 94
- S-atribuída, 132
- session storage, 29
- shortcut, 33
- siguientes, 97
- SLR, 99, 100
- static link, 108
- tabla de acciones, 99
- tabla de gotos, 99
- tabla de saltos, 99
- target phase, 43
- text area, 29
- Web storage, 29

Bibliografía

- [1] Mark Pilgrim. *Dive into HTML5*. <http://diveinto.html5doctor.com/index.html>, 2013.
- [2] G. Wilson and A. Oram. *Beautiful Code: Leading Programmers Explain How They Think*. O'Reilly Media, 2008.
- [3] Casiano Rodriguez Leon. *Perl: Fundamentos*. <http://nereida.deioc.ull.es/~lhp/perlexamples/>, 2001.
- [4] Nathan Whitehead. *Create Your Own Programming Language*. <http://nathansuniversity.com/>. 2012.
- [5] Nathan Whitehead. *What's a Closure?*. <http://nathansjslessons.appspot.com/>. 2012.
- [6] Steven S. Muchnick. *Advanced compiler design and implementation*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1997.
- [7] T. Mogensen and T.A. Mogensen. *Introduction to compiler design*. Undergraduate topics in computer science. Springer London, Limited, 2011.