



UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA
Escuela Técnica Superior de Ingeniería Informática



Procesadores de Lenguajes

Práctica de Procesadores de Lenguajes

Análisis Sintáctico con Cup

Javier Vélez Reyes
jvelez@lsi.uned.es

Javier Vélez Reyes jvelez@lsi.uned.es

Objetivos

- Aprender a construir analizadores sintacticos Cup
- Aprender a especificar gramaticas en Cup
- Aprender a trabajar con atributos en Cup
- Aprender a gestionar errores en Cup
- Aprender a configurar el analizador

Índice

- Instalación
- Introducción
- Especificación
- Traducción dirigida por la sintaxis
- Recuperación de errores
- Configuración

Instalación

- Instalación de Java
 - Descargar JDK de la página Web de Sun
 - Instalar JDK
 - Crear una variable de entorno CLASSPATH
 - Crear una variable de entorno JAVA_HOME a /bin
- Instalación de Cup
 - Descargar de <http://garoe.lsi.uned.es/procleng/practica>
 - Crear directorio de trabajo /pdl
 - Crear directorio para Cup /pdl/java_cup
 - Copiar en /pdl/java_cup el contenido de /java_cup
 - Compila
 - javac /pdl/java_cup/*.java
 - javac /pdl/java_cup/runtime/*.java

Índice

- Instalación
- Introducción
 - ¿Qué es Cup?
 - Uso de Cup
 - Esquema de funcionamiento de Cup
- Especificación
- Traducción dirigida por la sintaxis
- Recuperación de errores
- Configuración

Introducción

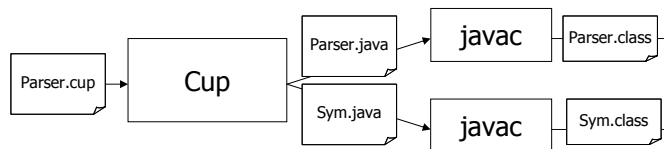
- Qué es Cup?
 - Cup es una herramienta para la construcción de analizadores sintácticos que genera parsers escritos en java (Constructor of Based Parsers)
 - Los parsers que se obtienen utilizan el método de análisis ascendente LALR
 - Hereda de YACC (versión en C)
 - Tiene una fácil y cómoda integración con la herramienta de análisis léxicos JLex

Introducción

- Uso de Cup
 - Especificar parser.cup en Cup
 - Lanzar Cup a ejecución con parser.cup

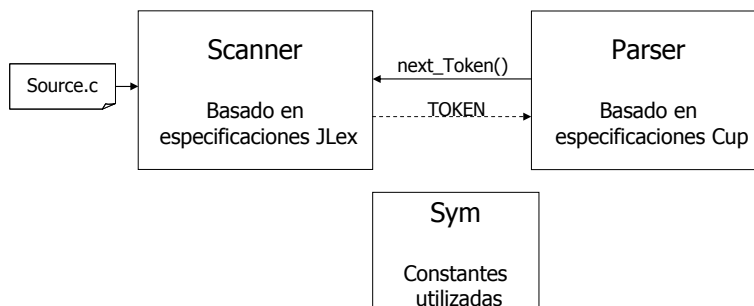
```
java java_cup.Main parser.cup
```

- Compilar parser.java y sym.java
- Interpretar parser.class



Introducción

- Esquema de funcionamiento Cup
 - El analizados sintáctico arranca
 - Solicita un token
 - El scanner lo devuelve de acuerdo a la codificación Sym



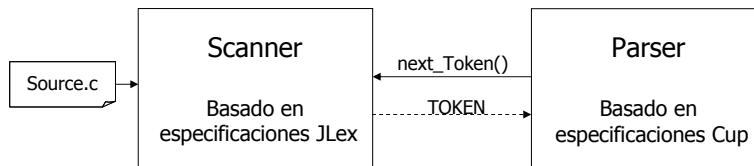
Introducción

- Esquema de funcionamiento de Cup
 - La clase Sym contiene una constante para cada token

```
package java_cup.simple_calc;

public class sym {
    /* terminals */
    public static final int SEMI = 2;
    public static final int EOF = 0;
    public static final int DIVIDE = 6;
    public static final int NUMBER = 11;
    public static final int error = 1;
    public static final int UMINUS = 8;
    public static final int MINUS = 4;
    public static final int TIMES = 5;
    public static final int LPAREN = 9;
    public static final int RPAREN = 10;
    public static final int MOD = 7;
    public static final int PLUS = 3;
}
```

Sym
Constantes
utilizadas

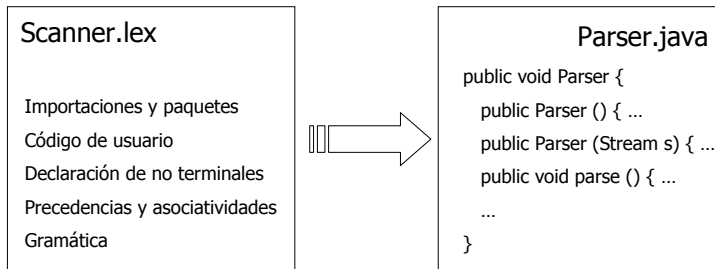


Índice

- Instalación
- Introducción
- Especificación
 - Importaciones
 - Código de usuario
 - Declaración de símbolos gramaticales
 - Asociatividad y precedencia
 - Gramática
- Traducción dirigida por la sintaxis
- Recuperación de errores
- Configuración

Especificación

- 5 partes
 - Especificación de paquete e importaciones
 - Código de usuario
 - Declaración de símbolos gramaticales
 - Declaración de precedencia y asociatividad
 - Gramática



Especificación

- Paquetes e importaciones
 - Paquete. Permite especificar el paquete java donde se desarrolla el proyecto del compilador

```
package name;
```

- Importaciones. Permite importar las clases necesarias que serán utilizadas dentro del analizador sintáctico

```
import package_name.class_name;

import package_name.*;
```

Especificación

■ Paquetes e importaciones

■ Ejemplo

```
package myPackage;
import java_cup.runtime.*;

/* Preliminaries to set up and use the scanner. */
init with (: scanner.init(); :);
scan with (: return scanner.next_token(); :);

/* Terminals (tokens returned by the scanner). */
terminal SEMI, PLUS, MINUS, TIMES, DIVIDE, MOD;
terminal UMINUS, LPAREN, RPAREN;
terminal Integer NUMBER;

/* Non terminals */
non terminal expr_list, expr_part;
non terminal Integer expr, term, factor;

/* Precedences */
precedence left PLUS, MINUS;
precedence left TIMES, DIVIDE, MOD;
precedence left UMINUS;

/* The grammar */
expr_list ::= expr_list expr_part |
           expr_part;
expr_part ::= expr SEMI;
expr      ::= expr PLUS expr
           | expr MINUS expr
           | expr TIMES expr
           | expr DIVIDE expr
           | expr MOD expr
           | MINUS expr %prec UMINUS
           | LPAREN expr RPAREN
           | NUMBER
           ;
```

Especificación

■ Código de usuario

- Action Code. Opcional. Permite definir variables y procedimientos de asistencia al parser dentro de una clase de ayuda

action code { : ... : }

- Parser Code. Opcional. Permite definir funciones y variables dentro de la clase Parser para adaptar el comportamiento del analizador sintáctico

parser code { : ... : }

Especificación

■ Código de usuario

- Init With. Opcional. Permite definir lógica de inicialización que se llamará antes de comenzar el análisis

```
init with { : ... : }
```

- Scan With. Opcional. Permite indicar qué sentencia debe utilizarse para solicitar un nuevo token

```
scan with { : ... : }
```

Especificación

■ Código de usuario

■ Ejemplo

```
package myPackage;
import java_cup.runtime.*;

/* Preliminaries to set up and use the scanner. */
init with { : scanner.init(); : };
scan with { : return scanner.next_token(); : };

/* Terminals (tokens returned by the scanner). */
terminal SEMI, PLUS, MINUS, TIMES, DIVIDE, MOD;
terminal UMINUS, LPAREN, RPAREN;
terminal Integer NUMBER;

/* Non terminals */
non terminal expr_list, expr_part;
non terminal Integer expr, term, factor;

/* Precedences */
precedence left PLUS, MINUS;
precedence left TIMES, DIVIDE, MOD;
precedence left UMINUS;

/* The grammar */
expr_list ::= expr_list expr_part |
            expr_part;
expr_part ::= expr SEMI;
expr      ::= expr PLUS expr
            | expr MINUS expr
            | expr TIMES expr
            | expr DIVIDE expr
            | expr MOD expr
            | MINUS expr %prec UMINUS
            | LPAREN expr RPAREN
            | NUMBER
            ;
```


Especificación

- Declaración de símbolos gramaticales
 - Terminales. Se especifican los símbolos terminales de la gramática indicando, opcionalmente, los tipos

```
terminal classname name1, name2, ...  
terminal name1, name2, ...
```

- No terminales. Se especifican los símbolos no terminales de la gramática indicando, opcionalmente, sus tipos

```
non terminal classname name1, name2, ...  
non terminal name1, name2, ...  
nonterminal name1, name2, ...  
nonterminal classname name1, name2, ...
```

Especificación

- Declaración de símbolos gramaticales
 - Las palabras reservadas no pueden utilizarse como nombres de terminales o no terminales
 - Son palabras reservadas en Cup

```
Code    action  parser  terminal    non  
nonterminal  init   scan   with    start  
precedence  left   right  nonassoc  
import and   package
```

Especificación

■ Declaración de símbolos gramaticales

■ Ejemplo

```
package myPackage;
import java_cup.runtime.*;

/* Preliminaries to set up and use the scanner. */
init with (: scanner.init();           :);
scan with (: return scanner.next_token(); :);

/* Terminals (tokens returned by the scanner). */
terminal      SEMI, PLUS, MINUS, TIMES, DIVIDE, MOD;
terminal      UMINUS, LPAREN, RPAREN;
terminal Integer  NUMBER;

/* Non terminals */
non terminal   expr_list, expr_part;
non terminal Integer  expr, term, factor;

/* Precedences */
precedence left PLUS, MINUS;
precedence left TIMES, DIVIDE, MOD;
precedence left UMINUS;

/* The grammar */
expr_list ::= expr_list expr_part |
            expr_part;
expr_part ::= expr SEMI;
expr      ::= expr PLUS expr
            | expr MINUS expr
            | expr TIMES expr
            | expr DIVIDE expr
            | expr MOD expr
            | MINUS expr %prec UMINUS
            | LPAREN expr RPAREN
            | NUMBER
            ;
```

Especificación

■ Asociatividad y precedencia

- Parte opcional que permite especificar explícitamente la asociatividad y precedencia de los terminales
- Resulta útil para soportar análisis con gramáticas ambiguas y resolver algunos conflictos reducción-desplazamiento
- Sintaxis

```
precedence left terminal1, terminal2, ...
precedence right terminal1, terminal2, ...
precedence nonassoc terminal1, terminal2, ...
```

Especificación

■ Asociatividad

```
precedence left terminal1, terminal2, ...  
precedence right terminal1, terminal2, ...  
precedence nonassoc terminal1, terminal2, ...
```

■ Tres posibles valores

- Left. Indica asociatividad a izquierdas
- Right. Indica asociatividad a derechas
- Nonassoc. Indica que el operador no es asociativo

Especificación

■ Precedencia

```
precedence left terminal1, terminal2, ...  
precedence right terminal1, terminal2, ...  
precedence nonassoc terminal1, terminal2, ...
```

- Indica la prioridad de evaluación de operadores
 - Los operadores en una misma fila tiene igual prioridad
 - La prioridad es creciente en orden de declaración

Especificación

■ Asociatividad y precedencia

■ Ejemplo

```
package myPackage;
import java_cup.runtime.*;

/* Preliminaries to set up and use the scanner. */
init with (: scanner.init(); :);
scan with (: return scanner.next_token(); :);

/* Terminals (tokens returned by the scanner). */
terminal SEMI, PLUS, MINUS, TIMES, DIVIDE, MOD;
terminal UMINUS, LPAREN, RPAREN;
terminal Integer NUMBER;

/* Non terminals */
non terminal expr_list, expr_part;
non terminal Integer expr, term, factor;

/* Precedences */
precedence left PLUS, MINUS;
precedence left TIMES, DIVIDE, MOD;
precedence left UMINUS;

/* The grammar */
expr_list ::= expr_list expr_part |
            expr_part;
expr_part ::= expr SEMI;
expr      ::= expr PLUS expr
            | expr MINUS expr
            | expr TIMES expr
            | expr DIVIDE expr
            | expr MOD expr
            | MINUS expr %prec UMINUS
            | LPAREN expr RPAREN
            | NUMBER
            ;
```

Especificación

■ Gramática

- Definición del Axioma. Es posible especificar el símbolo no terminal de que constituye el axioma de la gramática

```
start with noTerminal;
```

- Si se omite esta especificación se considera como axioma de la gramática aquel no terminal asociado a la regla de producción que aparece en primer lugar

Especificación

■ Gramática

- Definición de producciones. Cada producción es una regla que indica por que símbolo no terminal puede sustituirse una forma de frase en una reducción
 - Antecedente. Un no terminal
 - Consecuente. Una o varias formas de frase

```
A ::= A b B      A ::= ;  
    | A c C  
    | A b C  
    ;
```

Especificación

■ Gramática

- Declaración de precedencias contextuales
 - Las relaciones de precedencia entre terminales se establecen en el bloque anterior
 - La precedencia de una regla es igual a la precedencia del último terminal que aparece en la parte derecha de la regla

```
expr ::= expr MAS expr
```

- Si la regla no tiene terminales entonces la regla tiene una precedencia mínima

Especificación

■ Gramática

■ Declaración de precedencias contextuales

- Para alterar la precedencia asignada a una regla de manera explícita se utilizan declaraciones de precedencias contextuales a la regla

■ Ejemplo

```
precedence left PLUS, MINUS;  
precedence left TIMES, DIVIDE, MOD;  
precedence left UMINUS;
```

```
expr ::= MINUS expr %prec UMINUS
```

Índice

- Instalación
- Introducción
- Especificación
- Traducción dirigida por la sintaxis
 - Integración con JLex
- Recuperación de errores
- Configuración

Traducción dirigida por la sintaxis

- Gramáticas de atributos
 - Cup soporta gramáticas de atributos
 - Cada terminal contiene atributos
 - Cada no terminal contiene atributos
 - Algunos ejemplos de atributos
 - Terminales
 - Lexema
 - Línea y columna
 - Tipo, Traducción ...
 - No terminales
 - Tipo
 - Traducción...

Traducción dirigida por la sintaxis

- Gramáticas de atributos
 - Los atributos se encapsulan en un objeto
 - Debe declararse el tipo de ese objeto para
 - Terminales
 - No terminales

```
import java_cup.runtime.*;

/* Terminals (tokens returned by the scanner). */
terminal      SEMI, PLUS, MINUS, TIMES, DIVIDE, MOD;
terminal      UMINUS, LPAREN, RPAREN;
terminal Integer  NUMBER;

/* Non terminals */
non terminal Object    expr_list, expr_part;
non terminal Integer  expr;

/* Precedences */
precedence left PLUS, MINUS;
precedence left TIMES, DIVIDE, MOD;
precedence left UMINUS, LPAREN;

/* The grammar */
expr_list ::= expr_list expr_part
          |
          expr_part;
```

Traducción dirigida por la sintaxis

- Gramáticas de atributos
 - Se recomienda utilizar la clase `java_cup.runtime.Symbol`
 - Implementar una clase `Info`
 - Encapsular instancias de `Info` en `Symbol.value`

Symbol

```
package java_cup.runtime;

public int left, right;
public Object value;
public int sym;
public int parse_state;

public class Symbol {

    public Symbol(int id, int l, int r, Object o) {...}
    public Symbol(int id, Object o) {...}
    public Symbol(int id, int l, int r) {...}
    public Symbol(int sym_num) {...}
    Symbol(int sym_num, int state) {...}

    public String toString() {...}
}
```

```
public class Info {
    String lexema;
    int linea, tipo;
    StringBuffer trad;
    public Info () { ... }
    public String getLexema () { ... }
    public int getLinea () { ... }
    public int getTipo () { ... }
    public StringBuffer getTrad () { ... }
    ...
}
```

Traducción dirigida por la sintaxis

- Gramáticas de atributos
 - ¿Cómo afecta esto a JLex?
 - %cup
 - Construir tokens mediante

```
<YYINITIAL> "," { info = new Info (...);
                    return new Symbol (Sym.X, info) }
```

```
package java_cup.runtime;

public int left, right;
public Object value;
public int sym;
public int parse_state;

public class Symbol {

    public Symbol(int id, int l, int r, Object o) {...}
    public Symbol(int id, Object o) {...}
    public Symbol(int id, int l, int r) {...}
    public Symbol(int sym_num) {...}
    Symbol(int sym_num, int state) {...}

    public String toString() {...}
}
```


Traducción dirigida por la sintaxis

- Gramáticas de atributos
 - ¿Cómo se propaga el valor de los atributos?
 - En cada reducción es posible construir y propagar hacia el antecedente un nuevo objeto del tipo especificado en la lista de no terminales
 - Se utiliza RESULT para representar el valor del antecedente

```
/* Non terminals */
non terminal Object      expr_list, expr_part;
non terminal Integer    expr;

/* Precedences */
precedence left PLUS, MINUS;
precedence left TIMES, DIVIDE, MOD;
precedence left UNINUS, LPAREN;

/* The grammar */
expr_list ::= expr_list expr_part
          |
          expr_part;

expr_part ::= expr:e
          (: System.out.println(" " + e); :)
          SEMI
          ;

expr      ::= expr:e1 PLUS expr:e2
          (: RESULT = new Integer(e1.intValue() + e2.intValue()); :)
          ;
```

Traducción dirigida por la sintaxis

- Gramáticas de atributos
 - ¿Cómo se accede a los atributos desde Cup?
 - Cada símbolo gramatical puede etiquetarse con un nombre de variable que contiene el objeto encapsulado dentro de Symbol.value

```
package java_cup.simple_calc;

import java_cup.runtime.*;

/* Terminals (tokens returned by the scanner). */
terminal SEMI, PLUS, MINUS, TIMES, DIVIDE, MOD;
terminal UNINUS, LPAREN, RPAREN;
terminal Integer NUMBER;

/* Non terminals */
non terminal Object      expr_list, expr_part;
non terminal Integer    expr;

/* Precedences */
precedence left PLUS, MINUS;
precedence left TIMES, DIVIDE, MOD;
precedence left UNINUS, LPAREN;

/* The grammar */
expr_list ::= expr_list expr_part
          |
          expr_part;

expr_part ::= expr:e
          (: System.out.println(" " + e); :)
          SEMI
          ;

expr      ::= expr:e1 PLUS expr:e2
          (: RESULT = new Integer(e1.intValue() + e2.intValue()); :)
          ;
```

Índice

- Instalación
- Introducción
- Especificación
- Traducción dirigida por la sintaxis
- Recuperación de errores
 - Recuperación de errores
 - Funciones de información de errores recuperables
 - Funciones de información de errores irrecuperables
- Configuración

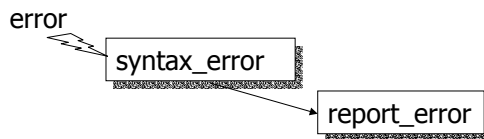
Recuperación de errores

- Recuperación de errores
 - Es posible definir las producciones de la gramática de manera que se recupera automáticamente de situaciones de error
 - Existe un símbolo no terminal especial error
 - Se añaden producciones de error
 - Ejemplo
 - Si la entrada no 'encaja' con ninguna de las producciones entonces se consumen tokens hasta encontrar ';' y se reduce

```
sent ::= expr PTOYCOMA
      | while PTOYCOMA
      | if PTOYCOMA
      | error PTOYCOMA
```

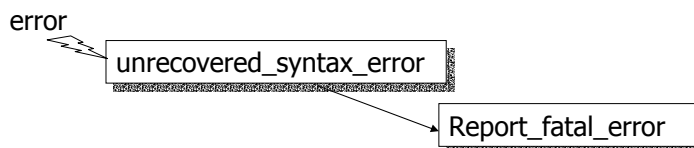
Recuperación de errores

- Funciones de información de errores recuperables
 - public void report_error (String message, Object info)
 - Debe sobrescribirse para tratar el error sintáctico y generar el mensaje de información adecuado a partir de los parámetros de entrada
 - public void syntax_error (Symbol cur_token)
 - Se llama cada vez que se da un error recuperable
 - Por defecto llama a report_error ("syntax error", null)
 - Sobrescribir a report_error ("syntax error", cur_token)



Recuperación de errores

- Funciones de información de errores irrecuperables
 - public void report_fatal_error (String message, Object info)
 - Debe sobrescribirse para tratar el error irrecuperable y generar el mensaje de información adecuado a partir de los parámetros de entrada
 - public void unrecovered_syntax_error (Symbol cur_token)
 - Se llama cada vez que se da un error irrecuperable
 - Por defecto llama a report_fatal_error ("syntax error", null)
 - Sobrescribir a report_fatal_error ("syntax error", cur_token)



Índice

- Instalación
- Introducción
- Especificación
- Traducción dirigida por la sintaxis
- Recuperación de errores
- Configuración

Configuración

- Opciones de ejecución
 - -package name. Especifica el paquete del proyecto
 - -parser name. Especifica el nombre del parser
 - -symbols name. Indica el nombre de la clase de símbolos
 - -interface. Declara la clase de símbolos como interfaz
 - -nonterminals. Añade no terminales a clase de símbolos
 - -expect n. Resuelve automáticamente n conflictos
 - -compact_red. Compacta la gestión de errores
 - -nowarn. Elimina el informe de warnings
 - -nosummary. Elimina el resumen del análisis sintáctico
 - -progress. Activa el informe de progreso

Configuración

- Opciones de ejecución
 - -dump_grammar. Volcado en pantalla de la gramática
 - -dump_states. Volcado en pantalla de los estados
 - -dump_tables. Volcado en pantalla de la tabla del parser
 - -dump. Todas las anteriores
 - -time. Muestra estadísticas de tiempo
 - -debug. Muestra información de debug
 - -noscanner. Ofrece compatibilidad con versiones antiguas
 - -version. Muestra la versión de Cup

Bibliografía

- [AJO] AHO, SETHI, ULLMAN: *Compiladores: Principios, técnicas y herramientas*; Addison-Wesley Iberoamericana, 1990



- [GARRIDO] A. Garrido, J. Iñesta, F. Moreno y J. Pérez. 2002. *Diseño de compiladores*. Universidad de Alicante.





UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA
Escuela Técnica Superior de Ingeniería Informática



Procesadores de Lenguajes

Práctica de Procesadores de Lenguajes

Análisis Léxico con JLex

Javier Vélez Reyes
jvelez@lsi.uned.es