Proyecto 2: JLex/CUP

Garay, Iñaki LU 67387; Touceda, Tomás LU 84024

13 de octubre de 2011

Proyecto 2: JLex/CUP
by Garay, Iñaki LU 67387; Touceda, Tomás LU 84024

Índice general

Capítulo 1

Gramatica completa del Mini-Lenguaje

```
<programa>              ::= <lista_sentencias> SCOLON EXIT
<lista_sentencias>      ::= <sentencia>
<lista_sentencias>      ::= <lista_sentencias> SCOLON <sentencia>
<sentencia>             ::= ID ASSIGNMENT <expression>
<sentencia>             ::= <sentencia_dib>
<sentencia>             ::= CLEAR
<sentencia>             ::= SETCOLOR <expression> COMMA <expression> COMMA < ↩
    expression>
<sentencia>             ::= IF PAREN_OPEN <expression> PAREN_CLOSE <sentencia_dib>
<sentencia>             ::= IF PAREN_OPEN <expression> PAREN_CLOSE <sentencia_dib> ↩
     ELSE <sentencia_dib>
<sentencia>             ::= REPEAT PAREN_OPEN <expression> PAREN_CLOSE TIMES < ↩
    sentencia_dib>
<sentencia_dib>         ::= DRAW <expression>
<sentencia_dib>         ::= MOVE <expression>

<expression>            ::= <expression> ADD <expression>
<expression>            ::= <expression> SUB <expression>
<expression>            ::= <expression> MUL <expression>
<expression>            ::= <expression> DIV <expression>
<expression>            ::= <expression> CONDITIONAL_AND <expression>
<expression>            ::= <expression> CONDITIONAL_OR <expression>
<expression>            ::= <expression> LT <expression>
<expression>            ::= <expression> GT <expression>
<expression>            ::= <expression> LT_EQ <expression>
<expression>            ::= <expression> GT_EQ <expression>
<expression>            ::= <expression> EQUALS <expression>
<expression>            ::= <expression> NOT_EQUALS <expression>
<expression>            ::= ADD <expression>
<expression>            ::= SUB <expression>
<expression>            ::= NOT <expression>
<expression>            ::= PAREN_OPEN <expression> PAREN_CLOSE
<expression>            ::= <postfix_expression>

<postfix_expression>    ::= ID
<postfix_expression>    ::= INT_LITERAL ANGLE
<postfix_expression>    ::= INT_LITERAL
<postfix_expression>    ::= DEF PAREN_OPEN <lista_sentencias_dib> PAREN_CLOSE

<lista_sentencias_dib> ::= <sentencia_dib>
<lista_sentencias_dib> ::= <lista_sentencias_dib> SCOLON <sentencia_dib>
```

Capítulo 2

Componentes lexicos

Token	Expresión regular	Ejemplos
IDENTIFIER	[a-zA-Z_\$][a-zA-Z_\$0-9]*	hola
WHITESPACE	[\t\r\n\f]+	
COMMENT	"/"[.]"*/"	/* comentario */
INT_LITERAL	[0-9]+	42
ID	[a-zA-Z][a-zA-Z0-9]*	id12
EXIT	"EXIT"	
CLEAR	"CLEAR"	
SETCOLOR	"SETCOLOR"	
IF	"IF"	
ELSE	"ELSE"	
REPEAT	"REPEAT"	
TIMES	"TIMES"	
DRAW	"DRAW"	
MOVE	"MOVE"	
DEF	"DEF"	
ASSIGNMENT	"←"	
CONDITIONAL_OR	"||"	
CONDITIONAL_AND	"&&"	
EQUALS	"=="	
NOT_EQUALS	"!="	
LT_EQ	"⇐"	
GT_EQ	">="	
SCOLON	";"	
COMMA	","	
PAREN_OPEN	"("	
PAREN_CLOSE	")"	
LT	"<"	
GT	">"	
ADD	"+"	
SUB	"-"	
MUL	"*"	
DIV	"/"	
NOT	"!"	
ANGLE	"D"	

Nota: Todos los tokens que contienen carácteres alfabéticos, si bien aparecen en mayúsculas, son igualmente considerados si aparecen en minúscula. Es decir, DEF es igual a dEf, y a la vez es igual a def.

2.1. Descripción general de la aplicación

2.2. Clases auxiliares usadas

2.2.1. `OurSymbol.java`

Esta clase abstrae la representación interna de unsímbolo, separada de lo que proporciona la clase `java_cup.runtime.Symbol`. A la vez, los objetos de este tipo son utilizados como filas en la tabla de símbolos (SymbolTable).

2.2.2. `OurDrawer.java`

OurDrawer, por como se diagramó la ejecución de los comandos de dibujo en casos como el de sentencias condicionales, es una clase que tiene la capacidad de interpretar comandos simples. Estos comandos están formados de la siguiente manera: "<comando>,<número>", donde <comando> puede ser "draw", "move", o "rotate", y <número> puede ser únicamente un entero.

2.2.3. `SymbolTable.java`

Esta clase es la que abstrae el manejo de variables y facilita los checkeos semánticos.

2.3. Casos de prueba

Los casos de prueba están focalizados a corroborar que los checkeos semánticos se comportan como lo estipulado. Los checkeos de los módulos de análisis léxico y sintáctico están implícitos en ellos y en los casos de tests correctos proporcionados por la cátedra.

Capítulo 3

Definición Dirigida por Sintaxis

Por una cuestión de comodidad de lectura y de formato, se expresará la DDS no de la forma usual (tabla con dos columnas), sino que se pondrá por cada regla, la acción correspondiente en la linea a continuación de la misma.

Asignación La acción agrega una variable a la tabla de símbolos. Si el resultado del método `addVar` es 1, significa que la variable ya habia sido asignada, y el tipo es compatible. Si el resultado es 2, significa que la variable ya habia sido asignada con un valor de otro tipo, y se levanta una excepción.

```
sentencia ::= ID:id ASSIGNMENT expression:e
```

```
OurSymbol es = table.get(e);

int res = table.addVar(es.getType(), id, es.getValue());
if (res == 1)
    table.setValue(id, es.getValue());
else if (res == 2) {
    String errormessage = "Tipo incompatible, no se puede asignar un " +
                          es.getType() +
                          " a un " +
                          table.get(id).getType();
    throw new Exception(errormessage);
}
```

Evaluación de una sentencia de dibujo La acción correspondiente a una sentencia de dibujo es llamar al wrapper de la clase Drawer, que interpreta la representación interna de una sentencia de dibujo y la ejecuta, mediante el metodo `evaluate` de la clase `OurDrawer`.

```
sentencia ::= sentencia_dib:s
```

```
ourDrawer.evaluate(s);
```

Sentencia de dibujo CLEAR La acción llama al método `reset` de la clase `OurDrawer`, el cual a su vez llama a `reset` de la clase `Drawer`.

```
sentencia ::= CLEAR
```

```
ourDrawer.reset();
```

Sentencia de dibujo SETCOLOR La acción correspondiente a la sentencia `SETCOLOR` verifica que los tres argumentos sean de tipo entero, y si los tres chequeos pasan, llama al método `setColor` de la clase `OurDrawer`.

```
sentencia ::= SETCOLOR expression:r COMMA expression:g COMMA expression:b
```

```
OurSymbol res_r = table.get(r);
OurSymbol res_g = table.get(g);
OurSymbol res_b = table.get(b);
```

```
if (!(res_r.getType().equals("int"))) { throw new Exception("El 1er argumento ↩
    debe ser de tipo entero!"); }
if (!(res_g.getType().equals("int"))) { throw new Exception("El 2do argumento ↩
    debe ser de tipo entero!"); }
if (!(res_b.getType().equals("int"))) { throw new Exception("El 3er argumento ↩
    debe ser de tipo entero!"); }
ourDrawer.setColor(res_r.getValue(), res_g.getValue(), res_b.getValue());
```

Sentencia condicional IF El tipo booleano es manejado internamente por el interprete como de tipo entero, al modo que se hace en C. El valor 0 es interpretado como Falso, y cualquier valor no-nulo como Verdadero.

La acción correspondiente a la sentencia condicional verifica que el tipo del resultado de la expresion condicional sea de tipo entero, y si es distinto de 0 evalua el resultado.

```
sentencia ::= IF PAREN_OPEN expression:econd PAREN_CLOSE sentencia_dib:thene
```

```
OurSymbol cond_sym = table.get(econd);
int cond_val = table.get(econd).getValue();

if (cond_sym.getType() == "int") {
    if (cond_val != 0) {
        ourDrawer.evaluate(thene);
    }
}
else {
    throw new Exception("Error! El tipo del resultado de la expresion condicional ↩
        debe ser entero o booleano.");
}
```

Sentencia condicional IF-ELSE

```
sentencia ::= IF PAREN_OPEN expression:econd PAREN_CLOSE sentencia_dib:thene ELSE ↩
    sentencia_dib:elsee
```

```
OurSymbol cond_sym = table.get(econd);
int condition = cond_sym.getValue();

if (cond_sym.getType().equals("int")) {
    if (condition != 0) {
        ourDrawer.evaluate(thene);
    }
    else {
        ourDrawer.evaluate(elsee);
    }
}
else {
    throw new Exception("Error! El tipo del resultado de la expresion condicional ↩
        debe ser entero o booleano.");
}
```

Sentencia de repetición REPEAT En el caso de la sentencia de repetición, el resultado de la expresión es interpretado como un valor numerico. El cuerpo de la sentencia de repetición se evalua tantas veces como indique el resultado de la expresión.

```
sentencia ::= REPEAT PAREN_OPEN expression:econd PAREN_CLOSE TIMES sentencia_dib: ↩
    rbody
```

```
OurSymbol cond_sym = table.get(econd);
int i = 0, times = cond_sym.getValue();

if (cond_sym.getType().equals("int")) {
    for (i = 0; i < times; i++) {
        ourDrawer.evaluate(rbody);
    }
```

```
}
else {
    throw new Exception("Error! El tipo del resultado de la expresion de  ↩
        repetición debe ser entero.");
}
```

Sentencia de dibujo DRAW La acción verifica que el tipo del argumento sea o bien entero o bien dibujo, y retorna una representación correspondiente a la sentencia a ejecutar. Esta representación es luego interpretada por el metodo `evaluate` cuando se parsee la regla que produjo esta sentencia.

Si el tipo del argumento es entero, entonces la representación sera un string de la forma `"draw,-<argumento>"`, donde `<argumento>` es el valor del argumento parseado, recuperado de la tabla de simbolos.

Si el tipo del argumento es dibujo, entonces la representación sera un string de la forma `"draw-_named,<identificador dibujo>"`. El valor del identificador se recupera tambien de la table de simbolos.

```
sentencia_dib ::= DRAW expression:e
```

```
/*
draw, int = avazar el cursor ang unidades
draw, did = reproducir un dibujo almacenado
*/

/* go see the eval method in the OurDrawer class. */
OurSymbol res = table.get(e);
if (res.getType() == "int") {
    RESULT = "draw," + res.getValue().toString();
}
else if (res.getType() == "draw_id") {
    RESULT = "draw_named," + res.getValue().toString();
}
else {
    throw new Exception("Error! El tipo del argumento de DRAW debe ser o bien  ↩
        entero o bien dibujo.");
}
```

Sentencia de dibujo MOVE La acción verifica que el operando sea o bien de tipo entero o bien de tipo angulo.

```
sentencia_dib ::= MOVE expression:e
```

```
/*
move, int = avanzar el cursor ang unidades sin dibujar
move, ang = rotar el cursor
*/

OurSymbol res = table.get(e);
if (res.getType() == "int") {
    RESULT = "move," + res.getValue().toString();
}
else if (res.getType() == "angle") {
    RESULT = "rotate," + res.getValue().toString();
}
else {
    throw new Exception("Error! El tipo del argumento de MOVE debe ser o bien  ↩
        entero o bien angulo.");
}
```

3.1. Expresiones

Suma Esta acción verifica que los tipos de los dos operandos de una suma sean iguales. Para cada subexpresión, se recupera de la tabla de símbolos la variable anonima que contiene su valor, y de ahi se

recupera su tipo. En caso de que difieran, se levanta una excepción. Caso contrario, se inserta el valor de la suma en una variable anonima en la table de simbolos.

```
expression ::= expression:e1 ADD expression:e2
```

```
OurSymbol e1s = table.get(e1);
OurSymbol e2s = table.get(e2);

if (e1s.getType() != e2s.getType())
    throw new Exception("Error: los operandos de una suma deben ser del mismo tipo.");

/* Este chequeo se haca despues porque ahora puedo asegurar que ambos
 * operandos son del mismo tipo, y puedo fijarme solo en el primero.
 */
if (!((e1s.getType() == "int") || (e1s.getType() == "angle")))
    throw new Exception("Error: los operandos de una suma deben ser o bien de tipo entero o bien de tipo angulo.");

String type = e1s.getType();
RESULT = table.addAnonymVar(type, e1s.getValue() + e2s.getValue());
```

Resta Esta acción realiza lo mismo para la operación de sustracción.

```
expression ::= expression:e1 SUB expression:e2
```

```
OurSymbol e1s = table.get(e1);
OurSymbol e2s = table.get(e2);

if (e1s.getType() != e2s.getType())
    throw new Exception("Error: los operandos deben ser del mismo tipo.");
if ((e1s.getType() != "int") && (e1s.getType() != "angle"))
    throw new Exception(e1s.getType() + " " + e2s.getType() + " Error: los operandos de una resta deben ser o bien de tipo entero o bien de tipo angulo.");

String type = e1s.getType();
RESULT = table.addAnonymVar(type, e1s.getValue() - e2s.getValue());
```

Multiplicación en una expresión

```
expression ::= expression:e1 MUL expression:e2
```

```
OurSymbol e1s = table.get(e1);
OurSymbol e2s = table.get(e2);

if ((e1s.getType() == e2s.getType()) && e1s.getType() == "angle")
    throw new Exception("Error: los operandos no pueden ser ambos de tipo angulo. ");

String type = "int";

if (e1s.getType() == "angle" || e2s.getType() == "angle")
    type = "angle";

RESULT = table.addAnonymVar(type, e1s.getValue() * e2s.getValue());
```

División en una expresión

```
expression ::= expression:e1 DIV expression:e2
```

```
OurSymbol e1s = table.get(e1);
OurSymbol e2s = table.get(e2);
```

```
if ((e1s.getType() == e2s.getType()) && e1s.getType() == "angle")
   throw new Exception("Error: los operandos no pueden ser ambos de tipo angulo." ↩
       );

if ((e1s.getType() != e2s.getType()) && e1s.getType() != "angle")
   throw new Exception("Error: solo el primero de los operandos puede ser de tipo ↩
        angulo.");

String type = "int";

if (e1s.getType() == "angle")
   type = "angle";

RESULT = table.addAnonymVar(type, e1s.getValue()/e2s.getValue());
```

Comparación por menor en una expresión

```
expression ::= expression:e1 LT expression:e2
```

```
OurSymbol e1s = table.get(e1);
OurSymbol e2s = table.get(e2);

if ((e1s.getType() == "draw_id") || (e2s.getType() == "draw_id"))
   throw new Exception("Error: ninguno de los operandos puede ser de tipo dibujo. ↩
       ");
if (e1s.getType() != e2s.getType())
   throw new Exception("Error: los operandos deben ser del mismo tipo.");

RESULT = table.addAnonymVar("int", (e1s.getValue() < e2s.getValue()) ? 1 : 0);
```

Comparación por mayor en una expresión

```
expression ::= expression:e1 GT expression:e2
```

```
OurSymbol e1s = table.get(e1);
OurSymbol e2s = table.get(e2);

if ((e1s.getType() == "draw_id") || (e2s.getType() == "draw_id"))
   throw new Exception("Error: ninguno de los operandos puede ser de tipo dibujo. ↩
       ");
if (e1s.getType() != e2s.getType())
   throw new Exception("Error: los operandos deben ser del mismo tipo.");

RESULT = table.addAnonymVar("int", (e1s.getValue() > e2s.getValue()) ? 1 : 0);
```

Comparación por menor o igual en una expresión

```
expression ::= expression:e1 LT_EQ expression:e2
```

```
OurSymbol e1s = table.get(e1);
OurSymbol e2s = table.get(e2);

if ((e1s.getType() == "draw_id") || (e2s.getType() == "draw_id"))
   throw new Exception("Error: ninguno de los operandos puede ser de tipo dibujo. ↩
       ");
if (e1s.getType() != e2s.getType())
   throw new Exception("Error: los operandos deben ser del mismo tipo.");

RESULT = table.addAnonymVar("int", (e1s.getValue() >= e2s.getValue()) ? 1 : 0);
```

Comparación por mayor o igual en una expresión

```
expression ::= expression:e1 GT_EQ expression:e2
```

```
OurSymbol e1s = table.get(e1);
OurSymbol e2s = table.get(e2);

if ((e1s.getType() == "draw_id") || (e2s.getType() == "draw_id"))
   throw new Exception("Error: ninguno de los operandos puede ser de tipo dibujo. ↩
       ");
if (e1s.getType() != e2s.getType())
   throw new Exception("Error: los operandos deben ser del mismo tipo.");

RESULT = table.addAnonymVar("int", (e1s.getValue() <= e2s.getValue()) ? 1 : 0);
```

Conjunción en una expresión condicional

```
expression ::= expression:e1 CONDITIONAL_AND expression:e2
```

```
OurSymbol e1s = table.get(e1);
OurSymbol e2s = table.get(e2);

if ((e1s.getType() == "draw_id") || (e2s.getType() == "draw_id"))
   throw new Exception("Error: ninguno de los operandos puede ser de tipo dibujo. ↩
       ");
if (e1s.getType() != e2s.getType())
   throw new Exception("Error: los operandos deben ser del mismo tipo.");
if (e1s.getType() != "int")
   throw new Exception("Error: los operandos no pueden ser de tipo angle.");

int val = 0;
if ((e1s.getValue().intValue() != 0) && (e2s.getValue().intValue() != 0))
    val = 1;

RESULT = table.addAnonymVar("int", val);
```

Disyunción en una expresión condicional

```
expression ::= expression:e1 CONDITIONAL_OR expression:e2
```

```
OurSymbol e1s = table.get(e1);
OurSymbol e2s = table.get(e2);

if ((e1s.getType() == "draw_id") || (e2s.getType() == "draw_id"))
   throw new Exception("Error: ninguno de los operandos puede ser de tipo dibujo. ↩
       ");
if (e1s.getType() != e2s.getType())
   throw new Exception("Error: los operandos deben ser del mismo tipo.");
if (e1s.getType() != "int")
   throw new Exception("Error: los operandos no pueden ser de tipo angle.");

int val = 0;

if (e1s.getValue().intValue() != 0 ||
   e2s.getValue().intValue() != 0)
    val = 1;

RESULT = table.addAnonymVar("int", val);
```

Comparación por igualdad en una expresión

```
expression ::= expression:e1 EQUALS expression:e2
```

```
OurSymbol e1s = table.get(e1);
OurSymbol e2s = table.get(e2);

if ((e1s.getType() == "draw_id") || (e2s.getType() == "draw_id"))
```

```
   throw new Exception("Error: ninguno de los operandos puede ser de tipo dibujo. ↩
       ");
if (e1s.getType() != e2s.getType())
   throw new Exception("Error: los operandos deben ser del mismo tipo.");

RESULT = table.addAnonymVar("int", (e1s.getValue().intValue() == e2s.getValue(). ↩
    intValue())?1:0);
```

Comparción por desigualdad en una expresión

```
expression ::= expression:e1 NOT_EQUALS expression:e2
```

```
OurSymbol e1s = table.get(e1);
OurSymbol e2s = table.get(e2);

if ((e1s.getType() == "draw_id") || (e2s.getType() == "draw_id"))
   throw new Exception("Error: ninguno de los operandos puede ser de tipo dibujo. ↩
       ");
if (e1s.getType() != e2s.getType())
   throw new Exception("Error: los operandos deben ser del mismo tipo.");

RESULT = table.addAnonymVar("int", (e1s.getValue().intValue() != e2s.getValue(). ↩
    intValue()) ? 1 : 0);
```

Suma unaria en una expresión

```
expression ::= ADD expression:e1
```

```
OurSymbol s = table.get(e1);
RESULT = table.addAnonymVar(s.getType(), + s.getValue());
```

Resta unaria en una expresión

```
expression ::= SUB expression:e1
```

```
OurSymbol s = table.get(e1);
RESULT = table.addAnonymVar(s.getType(), - s.getValue());
```

Negación en una expresión

```
expression ::= NOT expression:e1
```

```
OurSymbol s = table.get(e1);
Integer val = s.getValue();
if (val == 0)
    RESULT = table.addAnonymVar(s.getType(), 1);
else
    RESULT = table.addAnonymVar(s.getType(), 0);
```

3.2. Expresiones postfijas

Identificador

```
postfix_expression ::= ID:id
```

```
if (table.isDeclared(id))
    RESULT = id;
else
    throw new Exception("Variable no declarada: "+id);
```

Literal numerico (angulo) Guarda el valor del literal numerico de tipo angulo en una variable anonima en la tabla de simbolos.

```
postfix_expression ::= INT_LITERAL:lit ANGLE
```

```
RESULT = table.addAnonymVar("angle", lit);
```

Literal numerico (entero) Guarda el valor del literal numerico de tipo entero en una variable anonima en la tabla de simbolos.

```
postfix_expression ::= INT_LITERAL:lit
```

```
RESULT = table.addAnonymVar("int", lit);
```

Sentencia DEF

```
postfix_expression ::= DEF PAREN_OPEN lista_sentencias_dib:list PAREN_CLOSE
```

```
ourDrawer.setNamedDraw(last_draw_id, list);
RESULT = table.addAnonymVar("draw_id", last_draw_id++);
```