Revisión de herramientas de visualización de la Traducción Dirigida por Sintaxis

Damián Aníbal Nicolalde Rodríguez[1], Jaime Urquiza Fuentes[2]

[1] Pontificia Universidad Católica del Ecuador;
Av. 12 de Octubre 1076 y Roca, Quito, Ecuador
[2] Universidad Rey Juan Carlos;
Calle Tulipán, s/n, 28933 Móstoles, Madrid, España
da.nicolalde@alumnos.urjc.es; jaime.urquiza@urjc.es

Resumen: La enseñanza de la asignatura de Compiladores se ha convertido en un verdadero reto debido a la existencia de conceptos abstractos que no se comprenden inmediatamente. Pensando que la visualización ayudará a esta comprensión, este estudio analizará las soluciones publicadas para la visualización de la traducción dirigida por la sintaxis. Tras buscar en la bibliografía se identificaron las herramientas LISA, CUPV, VCOCO y EvDebugger. De estas se analizaron parámetros como el ámbito de uso, aspectos visualizados o la concordancia que existe entre la visualización y la especificación del traductor. El resultado más sorprendente es que la herramienta no diseñada para el ámbito educativo, LISA, presenta mayor concordancia entre la visualización y la especificación. Aparte de esto cabe destacar: la coherencia con la asignatura y concordancia entre especificación y visualización de EvDebugger, las posibilidades interactivas de LISA y el uso de notaciones estándar en LISA, CUPV y VCOCO.

Palabras clave: Visualización del software, Traducción dirigida por la sintaxis, Procesadores de Lenguajes, Compiladores.

Abstract: Teaching compilers has become a real challenge due to the existence of abstract concepts that are not understood immediately. Given that visualization can help understanding these concepts, this study aims to analyze works related to visualization of syntax directed translation. As a result of the literature review four tools were identified: LISA, CUPV, VCOCO and EvDebugger. The analysis has been carried out based on a set of parameters that will guide us in future works. The most surprising result is that the only tool not designed for education, LISA, is the one that is available for download and use. Apart from this, we can highlight the coherence with the subject and the sound mapping between specification and visualization of EvDebugger, the interactive possibilities of LISA and the use of standard notations in LISA, CUPV and VCOCO.

Key words: Software visualization, Syntax directed translation, Language processors, Compilers.

1. Introducción

Procesadores de lenguaje y compiladores son consideradas por los estudiantes como unas de las asignaturas más difíciles en los grados de Ciencias de la Computación. Un tema clave dentro de ellas es la traducción dirigida por la sintaxis. Existen múltiples herramientas que facilitan el desarrollo de traductores dirigidos por la sintaxis como ANTLR[1], BISON[2] o CUP[3] entre otras. Todas ellas tienen un claro enfoque

[1] http://www.antlr.org/
[2] https://www.gnu.org/software/bison/
[3] http://www2.cs.tum.edu/projects/cup/

profesional. El objetivo de esta revisión es describir los trabajos centrados en ayudar a la comprensión de la traducción dirigida por la sintaxis, especialmente aquellos que usan la visualización.

1.1. Enseñanza de materias complejas

La enseñanza de materias complejas como la matemática, programación, procesadores de lenguaje se ha convertido en un verdadero reto para el profesor, ya que son materias donde el estudiante tiene que desarrollar y aplicar el razonamiento abstracto, por lo que debe encontrar los recursos necesarios para motivar a los estudiantes. El método tradicional en que el docente imparte las clases (por lo general clase magistral) no logra tanto la motivación, ocasionando porcentajes elevados de deserción al finalizar los cursos.

En la actualidad, el uso de las Tecnologías de la Información y Comunicación (TIC) en la educación juegan un papel muy importante; su aplicación va a exigir la creación de nuevos modelos de aprendizaje, nuevos procedimientos y estrategias de búsqueda, organización, procesamiento y utilización de la información, estimulando a los docentes a enfocarse en el diseño de métodos sistemáticos que le impulsen al estudiante a aprender a aprender [1].

Aprovechando, las ventajas que las TIC presentan para el proceso de enseñanza – aprendizaje como el uso de simuladores, software especializado, visualización de software, etc., los centros de estudio están generando modelos que permitan la enseñanza de materias complejas. En las Ciencias de la Computación la materia de Compiladores es considerada como una de las más complicadas por la complejidad de los algoritmos que deben comprender los estudiantes.

Estudios previos [2] concluyen que al utilizar adecuadamente las TIC en la enseñanza de materias complejas como las matemáticas por ejemplo, se puede ayudar a mejorar aspectos motivacionales, actitudinales y académicos en los estudiantes. Debido a esto los docentes deben buscar estrategias para impartir sus clases con el fin de que sus estudiantes las puedan comprender. Entre las estrategias para la enseñanza de materias complejas está el uso de las TIC, en el caso de las Ciencias de la Computación y específicamente en la enseñanza de la asignatura de Compiladores se pueden usar técnicas de visualización de software para mostrar los conceptos de la Traducción Dirigida por Sintaxis de forma gráfica, animada e interactiva y se pueden considerar como "potentes herramientas que permiten afianzar conceptos, definiciones, algoritmos y procedimientos". [3] cita a (Gonzales Alberto, 2013, p.1).

1.2. Traducción dirigida por la sintaxis

Las tres fases clásicas de un Procesador de Lenguajes son: análisis léxico (AL), análisis sintáctico(AS) y traducción dirigida por la sintaxis (TDS). Tomando a los compiladores como ejemplo paradigmático de los procesadores de lenguajes, su mecanismo reside principalmente en la TDS, en el que el control lo lleva el AS y todas las demás fases están sometidas a él [3]. Esto quiere decir, que el AS va construyendo poco a poco el árbol sintáctico y cada vez que necesita un nuevo componente léxico para continuar con la construcción, se lo solicita al AL. Así el AS obtiene una cadena de tokens del AL y verifica que dicha cadena pueda generarse mediante la gramática para el lenguaje fuente [4].

En general podemos encontrar tres tipos de AS [4]: universal, descendente y ascendente. Sin embargo, los usados regularmente para compiladores son los tipos descendentes y ascendentes. Los tipos de analizadores descendentes construyen el árbol desde la raíz a las hojas, utilizan la familia de técnicas LL -LL(1), LL(k)- y aunque su implementación podría llegar a ser manual generalmente se usan generadores automáticos como la ya mencionada herramienta ANTLR. Los de tipo ascendente construyen el árbol desde las hojas hacia la raíz, utilizan la familia de técnicas LR -LR(0), SLR(1), LR(1) y LALR(1)- y su implementación manual es sumamente costosa por lo que se afronta con generadores automáticos como las herramientas también anteriormente mencionadas CUP o BISON. En ambos casos la cadena de componentes léxicos se procesa de izquierda a derecha [5].

La TDS consiste en usar la aplicación de reglas sintácticas, derivaciones en analizadores descendentes o reducciones en analizadores ascendentes, para ejecutar acciones de mayor complejidad. Además, estas acciones suelen usar elementos de información asociados a los símbolos gramaticales, llamados atributos. La TDS se suele especificar con gramáticas de atributos. Según [6], una gramática de atributos se puede definir como AG=(G, A, R) donde:

- G es una gramática libre de contexto G=(T, N, P, S); donde T es el conjunto de símbolos terminales, N es el conjunto de símbolos no terminales, S es el símbolo de inicio y es parte de N y P es el conjunto de producciones.
- A es un conjunto de atributos A(x) para cada símbolo no terminal x. A(x) es dividido en dos conjuntos mutuamente excluyentes, de atributos heredados y atributos sintetizados I(x) y S(x) respectivamente.
- R es un conjunto finito de reglas semánticas que permitirán especificar las acciones complejas a ejecutar durante el proceso de TDS.

En el ámbito de los compiladores la TDS da soporte para la comprobación de tipos y la generación de código intermedio. Un ejemplo clásico es la calculadora de escritorio [4] que mostramos en la figura 1. En esta figura se muestra la gramática, a la que se añaden las acciones (*en cursiva*) que principalmente permiten realizar los cálculos necesarios y los atributos (**en negrita**) que permiten comunicar los resultados parciales de cada cálculo así como el resultado final. Así, cuando el AS aplique la regla sintáctica `E ::= E1 + T`, el TDS ejecutará la suma (la acción asociada a la regla) usando como operandos los atributos `v` de los símbolos `E1` y `T`, y lo guardará en el atributo `v` del símbolo `E`.

```
S ::= E \n          { printf(E.v); }
E ::= E1 + T        { E.v = E1.v + T.v; }
       | E1 - T     { E.v = E1.v - T.v; }
       | T          { E.v = T.v; }
T ::= T1 * F        { T.v = T1.v * F.v; }
       | T1 / F     { T.v = T1.v / F.v; }
       | F          { T.v = F.v; }
F ::= ( E )         { F.v = E.v; }
       | constante  { F.v = constante.v; }
```

Figura 1. Ejemplo de calculadora de escritorio

Según esta descripción, los conceptos fundamentales que tienen que manejar los estudiantes son: los atributos -tanto la asignación de los valores como su utilización-, las acciones asociadas a las reglas sintácticas y su momento de ejecución (muy relacionado con el tipo de analizador sintáctico) Nuestro objetivo entonces será estudiar cómo las herramientas existentes apoyan la comprensión de estos conceptos.

1.3. Herramientas de visualización en la educación

El uso de la visualización como herramienta educativa no es una novedad, sirva como ejemplo el vídeo sobre algoritmos de ordenación creado por Baecker [7] en la década de los 90. Aunque la representación visual de conceptos abstractos puede ayudar a su comprensión, la eficacia educativa de las visualizaciones requiere de un análisis previo sobre el uso que los estudiantes harán de ella [8].

En el ámbito de los procesadores de lenguajes existen múltiples ejemplos de herramientas de visualización, casi todas relacionadas con la fase de análisis sintáctico tanto desde un punto de vista teórico, p.ej. JFLAP [9], como práctico, p.ej. VAST[10]. Se considera la visualización de la TDS puede mejorar la eficacia educativa, permitiendo al estudiante interactuar con las diferentes fases del proceso y simulando el comportamiento de las acciones semánticas en el árbol sintáctico para la evaluación de los atributos.

Este artículo presenta una revisión de los trabajos sobre visualización de la traducción dirigida por la sintaxis. En la sección 2 se describen los objetivos detallados y la metodología utilizada en esta revisión. La sección 3 presenta las herramientas revisadas. Finalmente, en la sección 4 se presentan las conclusiones de la revisión.

2. Objetivos y metodología de la revisión

Para la revisión se utilizó la metodología propuesta por Webster y Watson [11] en la que se identifican los siguientes pasos: definición del ámbito de la revisión, especificación de parámetros, identificación de la literatura relevante, análisis de la literatura relevante y extracción de conclusiones. Como ya se ha comentado en el artículo, el ámbito de esta revisión es la visualización de la TDS, el objetivo fundamental es identificar las brechas que dejan las herramientas dentro de este ámbito, de manera que se motive a los

investigadores para futuros trabajos.

Siguiendo la metodología antes mencionada, el segundo paso consiste en especificar los parámetros clave que guiarán el proceso de revisión. Se han identificado los siguientes siete parámetros:

i. Formato de las especificaciones.
ii. Fases de un procesador de lenguajes involucradas.
iii. Ámbito de uso.
iv. Aspectos visualizados.
v. Concordancia entre la especificación y la visualización.
vi. Disponibilidad.
vii. Evaluaciones de su uso.

En cuanto al *formato de las especificaciones*, es interesante saber si las especificaciones se escriben siguiendo un formato propietario, por ejemplo, el usado por un profesor concreto en sus clases, o totalmente basado en un estándar como BNF o EBNF, o un punto intermedio que se representaría por un formato no estándar pero sí ligado a una herramienta de generación previamente existente con mayor o menor difusión en el ámbito de uso. Una de las cuestiones que podrían estar influenciadas por este parámetro es la posibilidad de uso fuera del contexto en que fue diseñada la herramienta.

Con las *fases de un procesador de lenguajes involucradas* se identifican las fases que el usuario deberá tener en cuenta en el desarrollo de las especificaciones de traductores. Como es lógico podrían ser una o varias de las tres ya descritas: AL, AS o TDS.

El *ámbito de uso* podrá ser profesional o educativo. Para el primer caso la herramienta podrá ser utilizada para la generación automática de analizadores sintácticos y traductores y en el segundo caso podrá ser usada en entornos educativos durante el proceso de enseñanza-aprendizaje del AS y la TDS.

Con los *aspectos visualizados* se hace un análisis de las fases del compilador que permite visualizar la herramienta y cómo es la forma en que esta visualización se realiza. Por ejemplo, el AL se puede visualizar como un autómata o como los patrones que se van encontrando. El AS se puede visualizar como reglas aplicadas o como el propio árbol sintáctico creado e incluso se pueden añadir vistas de la pila o cualquier otro elemento de interés en el AS. Respecto al TDS se pueden visualizar múltiples aspectos como es la ejecución de acciones semánticas, el uso de los distintos atributos, sus valores en determinados momentos del análisis e incluso el grafo de dependencias. Finalmente, también interesa saber si la visualización es estática o dinámica (animación)

La *concordancia entre visualización y especificación* estudia la adaptación de la visualización al modelo mental del usuario. Si el modelo mental del usuario cuando produce una especificación trabaja con elementos como producción, atributo o acción semántica, la visualización deberá mostrar esos elementos de forma que las explicaciones visuales generadas sean más fácilmente entendibles por el usuario.

Desde un punto de vista global, la concordancia será total cuando la visualización está totalmente basada en la especificación, será parcial si la visualización está en parte basada en la especificación ya sea porque no muestra todos los elementos o todas las fases y nula, que significa que la visualización no se basa en la especificación, mostrando otros elementos visuales. Además, para determinar de forma más objetiva el grado de concordancia que existe entre la visualización y la especificación, se ha definido la escala numérica basada en tres aspectos: el nombre de los atributos y los valores que toman en el momento de la ejecución del traductor (referenciada posteriormente como AV), la creación visual del árbol sintáctico (referenciada posteriormente como ArS) y la información visual ofrecida sobre la acción semántica que se ejecuta en ese momento (referenciada posteriormente como ASn) a cada uno de estos tres aspectos les damos los de 0.3, 0.3 y 0.4, respectivamente. Se le ha dado mayor peso a la información visual sobre la acción semántica que se ejecuta en un momento determinado con un valor de 0.4, por estar directamente relacionada con la TDS.

En cuanto a la *disponibilidad*, se analiza la disponibilidad real de la herramienta: si está disponible y se puede usar o sólo está accesible su descripción en

un documento científico-técnico sin poder disponer de la herramienta para ejecutarla o simplemente existe alguna mención a la herramienta sin documentación ni software que utilizar.

Finalmente, con las *evaluaciones de su utilización* se analiza la existencia de informes sobre la evaluación de la utilización de la herramienta en el ámbito para el que se ha diseñado, así como el grado de formalidad en la evaluación, diferenciando entre diseños experimentales, quasi-experimentales o meramente informales.

3. Corpus de la revisión

El siguiente paso de la metodología de revisión es la identificación de la literatura relevante. Para ello se han usado las bibliotecas digitales de ACM[4] e IEEE[5] junto con dos tesis doctorales relacionadas con el ámbito del trabajo [12][13]. Como se dijo anteriormente el uso de la visualización no es una novedad, existen numerosos ejemplos de su utilización en el ámbito educativo. El hecho de que tanto compiladores como procesadores de lenguajes sean un campo muy concreto de la informática no es óbice para que existan variedad de herramientas de visualización en este campo. Sin embargo, no todas están relacionadas con la TDS, que es nuestro campo de estudio. Algunas de estas herramientas que no hemos incluido en nuestro análisis son JFLAP [9] más centrada en el aspecto teórico de autómatas, VAST [10] pensada para visualizar únicamente analizadores sintácticos LL y LR o SOTA [14] dedicada a la visualización de tablas de símbolos. Siguiendo la metodología propuesta hemos analizado cuatro herramientas: LISA, CUPV, VCOCO y EvDebugger.

3.1. LISA

LISA es un generador de compiladores/intérpretes interactivo compuesto de un conjunto de herramientas relacionadas tales como generadores de AL, generadores de AS, generadores de TDS, herramientas gráficas, herramientas de edición y conversión, que están integrados por interfaces bien diseñadas [15]. El sistema de construcción de LISA sigue un enfoque orientado a objetos y ya ha implementado mecanismos de herencia, modularidad y extensibilidad [14].

LISA parte de las especificaciones formales del lenguaje basado en la gramática de atributos [15]. Las partes léxicas y sintácticas de una especificación de lenguaje utilizan métodos formales bien conocidos, como expresiones regulares y BNF (Backus-Naur Form). Las semánticas se definen además con gramáticas de atributos similares a las mencionadas anteriormente. El lenguaje de especificaciones basado en la gramática de atributos en LISA tiene la estructura que se puede visualizar en la figura 2.

LISA es un sistema para generar analizadores LL, SLR, LALR y LR, además soporta presentaciones visuales de estructuras como autómatas de estado finito, BNF, árboles sintácticos, grafos de dependencia [17]. Al ser LISA un software desarrollado pensando en la productividad y no en el ámbito educativo, el propósito principal es la generación automática de compiladores a partir de las definiciones formales de lenguajes. La arquitectura de LISA se puede observar en (Fig. 3).

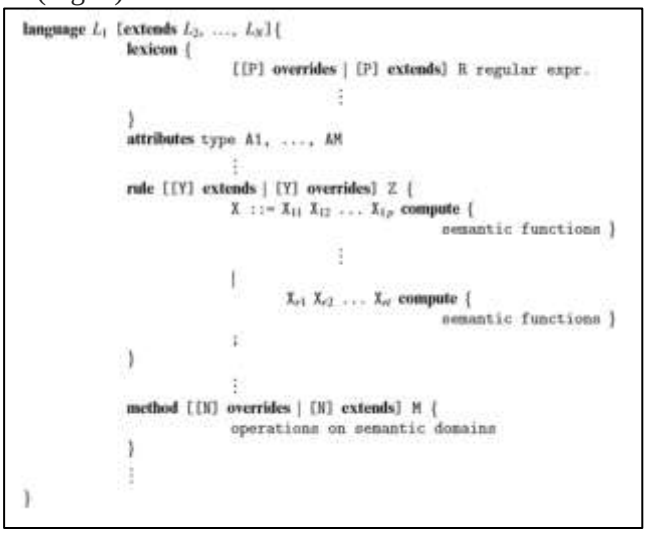

Figura 2. Estructura de un programa en LISA, [17]

[4] https://dl.acm.org/

[5] http://ieeexplore.ieee.org/

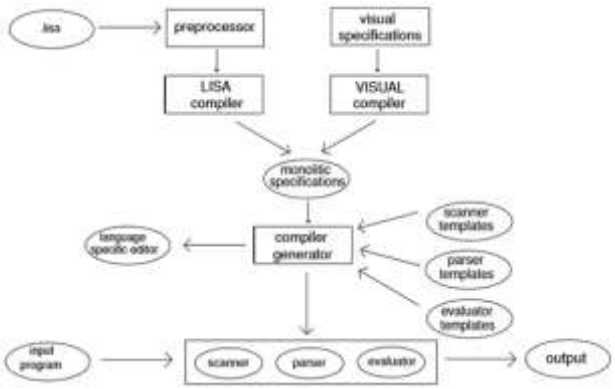

Figura 3. Arquitectura LISA, [16]

Los usuarios de LISA tienen la posibilidad de observar de forma visual el trabajo de los analizadores léxico, sintáctico y semántico, mediante la animación del autómata de estado finito, el análisis y la construcción del árbol semántico[18].
Como se puede observar en la especificación de la figura 4 las fases involucradas son el AL, AS y TDS.

Con la finalidad de evaluar el uso de LISA para mejorar la comprensión de la TDS, se analizó el funcionamiento y la manera en que presenta los resultados, haciendo énfasis en la visualización. Para lo cual se creó un lenguaje en LISA que permita evaluar expresiones aritméticas. Acorde con la estructura de la notación en LISA, la especificación del lenguaje se puede ver en la Fig. 4.

En el código mostrado en la figura 4, se puede observar la definición de la gramática, la definición de los atributos sintetizados y la definición de las reglas sintácticas. Una vez compilado y ejecutado el programa los resultados que muestra LISA son:

- Árbol de análisis sintáctico (Fig. 5): permite observar el árbol generado del lenguaje ejecutado.
- Árbol de evaluación (Fig. 6): permite observar en detalle el estado de cada nodo del árbol durante la evaluación, los valores actuales de los atributos heredados, así como el tipo.
- Grafo de dependencias (Fig. 7): muestra la dependencia entre los atributos de la gramática y también sirve como herramienta para detectar circularidad.

```
--ESCÁNER
language Trabajo {
    lexicon {
            Float [0-9]*.?[0-9]+
            Separator \( | \)
            Operator \+ | \- | \*
            ignore [\ \0x0D\0x0A\0x09]+
    }

--ATRIBUTOS
    attributes double *.val;

--REGLAS
    rule Exprecion1 {
            EXPR0 ::= EXPR compute {
            EXPR0.val = EXPR.val;};

            EXPR ::= TERM + TERM compute {
            EXPR.val = TERM[0].val + TERM[1].val;};

            EXPR ::= TERM - TERM compute {
            EXPR.val = TERM[0].val - TERM[1].val;};

            EXPR ::= TERM * TERM compute {
            EXPR.val = TERM[0].val * TERM[1].val;};
    }

    rule Terminal {
            TERM ::= #Float compute {
            TERM.val =
Double.valueOf(#Float.value()).doubleValue();};

            TERM ::= ( EXPR0 ) compute {
               TERM.val = EXPR0.val;};
    }
}
```

Figura 4. Ejemplo de especificación en LISA

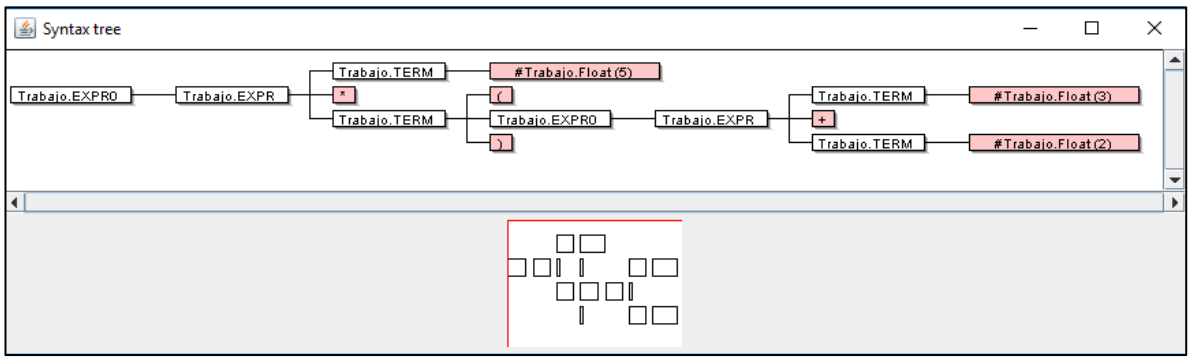

Figura 5. Árbol sintáctico

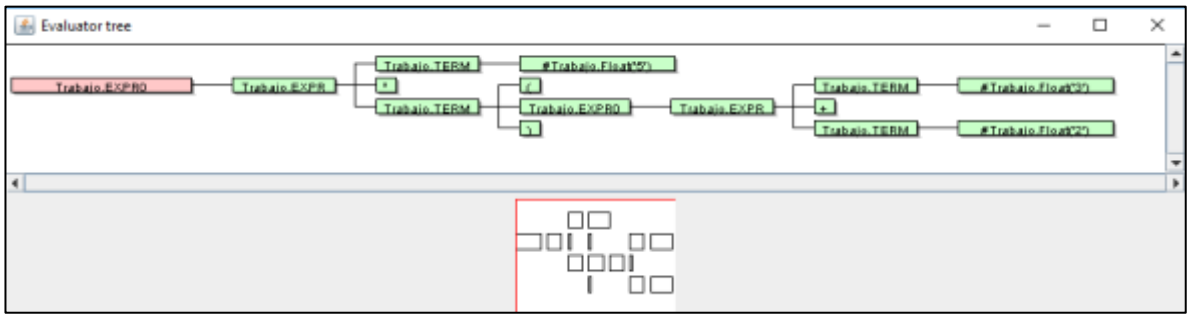

Figura 6. Árbol de evaluación

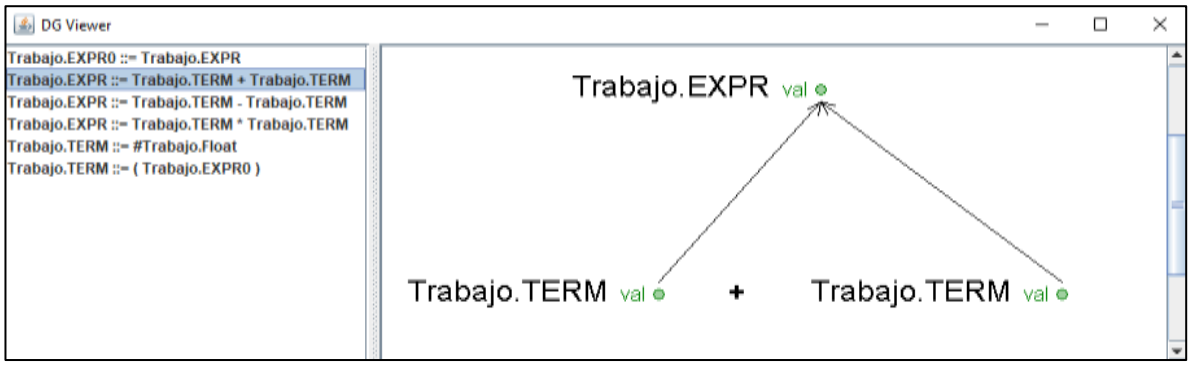

Figura 7. Grafo de dependencias

La concordancia entre la visualización y la especificación es total. La visualización está basada en la especificación del AS, permitiendo visualizar la construcción del árbol sintáctico dentro del árbol de evaluación. En lo que se refiere a TDS la herramienta es capaz de mostrar una concordancia entre los atributos que se manipulan y qué valores tienen estos en el árbol de evaluación, también muestra un grafo de dependencias local a cada producción y qué acciones de la especificación se están ejecutando en cada momento.
De acuerdo con la escala numérica definida en la metodología para establecer la concordancia existente, en la tabla 1, se puede observar que el resultado para LISA es de 1, ya que permite visualizar los atributos y los valores que toman en el momento de la ejecución del traductor, la creación visual del árbol sintáctico y la información sobre la acción semántica que se ejecuta en un momento determinado.

Tabla 1. Valoración numérica para LISA

Nivel de concordancia	LISA
AV	0.3
ArS	0.3
ASn	0.4
Total	**1**

En lo que se refiere a disponibilidad real de la herramienta, los autores si han podido encontrar una versión del software accesible para poder ejecutarla. Aunque la plataforma de despliegue hay que adaptarla a las condiciones en que esta herramienta se desarrolló (versiones antiguas de JDK, etc.) Por otro lado, no se ha encontrado documentación sobre evaluaciones educativas del uso de la herramienta.

3.2. CUPV

CUPV es una herramienta de visualización diseñada como una extensión de CUP, heredando todas sus características en lo que a especificaciones del AS y TDS se refiere, el formato utilizado es la notación BNF. En lo que respecta al AL, CUP delega su generación en herramientas externas como JFlex[6]. Permite visualizar los valores semánticos, las reducciones realizadas por el analizador, los estados el analizador, el árbol sintáctico y la entrada. Además, su diseño fue pensado para ayudar la comprensión de la ejecución del analizador por parte de los estudiantes permitiéndoles visualizar varios aspectos críticos del proceso de compilación [19], por lo que el ámbito de uso de esta herramienta es educativo.

Al ser CUPV una extensión de CUP, es un analizador sintáctico que utiliza la técnica LALR. El TDS está basado en reglas de producción, esto es, que las acciones semánticas pueden ser asociadas con las producciones de una gramática mediante la incrustación de código Java en la especificación de entrada, pudiendo tener el antecedente de atributos sintetizados. Tomando en cuenta la notación utilizada en CUP, en la figura 8, se muestra un fragmento de la especificación de entrada.

Cómo se puede observar en el fragmento de la especificación de la figura 8 se incluye las declaraciones para varios terminales y no terminales definidos por la gramática, los tipos atributos asociados con los terminales y no terminales también son incluidos, por lo que las fases involucradas son dos, AS y TDS.

```
/*Teminales */
terminal      SEMI, PLUS, MINUS, TIMES, DIVIDE;
terminal      MOD, UMINUS, LPREN, RPAREN;
terminal Integer NUMBER;

/*No Terminales*/
non terminal Object  expr_list, expr_part;
non terminal Integer expr;

/*Gramática*/
expr_list::=  expr_list expr_part
       |
       Expr_part;

expr_part::=  expr:a SEMI
       {:System.out.println("= " + a);:}

expr::=       expr:a1 PLUS expr:a2
       {:RESULT=new Integer(a1.intValue +
a2.intValue);:}
       |
       expr:a1 MINUS expr:a2
       {:RESULT=new Integer(a1.intValue -
a2.intValue);:}
       …;
```

Figura 8. Especificación en CUP, usada por CUPV, [19]

Una vez compilado y ejecutado el programa los resultados que muestra CUPV son:

[6] http://jflex.de/

- Reducciones realizadas por el analizador (Fig. 9): esta opción es usada para avisar a los usuarios cuando una reducción está a punto de ocurrir. Además, que muestra el valor de los atributos involucrados en la acción semántica que se ejecuta con la reducción.
- Visualización de los estados del AS (Fig. 10): el estado actual es visualizado como un botón etiquetado.
- Visualización del árbol sintáctico (Fig. 11): esta opción muestra el árbol sintáctico junto con el programa de entrada.

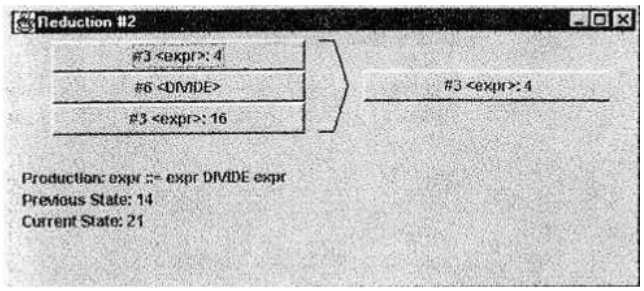

Figura 9. Pantalla de reducción [19]

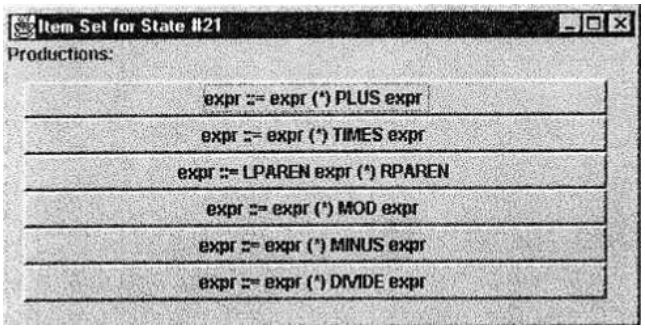

Figura 10. Visualización del conjunto de elementos para un estado [19]

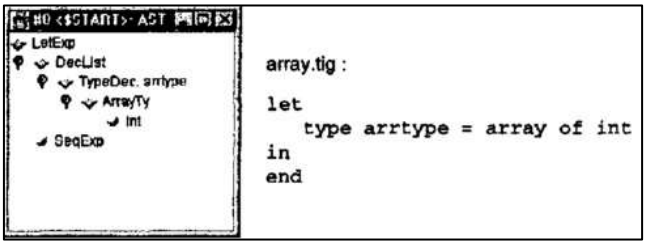

Figura 11. Visualización del árbol sintáctico y la entrada [19]

La concordancia entre la visualización y la especificación es parcial. La visualización está basada en parte en la especificación usando las producciones de la especificación. En lo que se refiere a TDS la herramienta es capaz de mostrar una concordancia entre los atributos que se manipulan y qué valores tienen estos, pero no muestra el árbol de dependencias ni las acciones semánticas ejecutadas indicando los atributos involucrados en ellas. En la tabla 2 se puede observar que el resultado de la valoración numérica para CUPV es de 0.6, ya que no permite visualizar la información sobre la acción semántica que se ejecuta en un momento determinado.

Tabla 2. Valoración numérica para CUPV

Nivel de concordancia	CUPV
AV	0.3
ArS	0.3
ASn	0
Total	**0.6**

En lo que se refiere a disponibilidad real de la herramienta, los autores no han podido encontrar una versión del software accesible para poder ejecutarla.
En [19] se hace referencia que CUPV se fue construido para usarlo en entornos educativos, sin embargo, no se han podido encontrar estudios donde se haga una evaluación educativa de la herramienta.

3.3. VCOCO

VCOCO [20] es una herramienta análoga a CUPV ya que representa una extensión del generador automático de parsers COCO/R[7], heredando por tanto todas sus características en lo que a especificaciones del AL, AS y TDS se refiere. Así, el formato utilizado es la notación EBNF tanto para el AL como para las reglas del AS, véase un ejemplo en la figura 12. El TDS se estructura siguiendo la idea básica vista en la sección 1.2. Es decir, cada producción tiene asociada un conjunto de acciones semánticas, pudiendo tener el antecedente atributos heredados y sintetizados. Como COCO/R es capaz de generar parsers escritos en diferentes lenguajes, la especificación de los atributos dependerá del lenguaje final elegido. Por ejemplo, en C# se permiten utilizar varios atributos sintetizados (también varios heredados) y sin embargo, en Java sólo se permite como máximo un atributo sintetizado por cada símbolo no terminal, pero al ser un objeto puede contener todos los atributos sintetizados necesarios dentro de él.

[7] http://www.ssw.uni-linz.ac.at/Coco/

```
COMPILER Calc

CHARACTERS
digit = '0' .. '9'.
TOKENS
number = digit {digit}.

PRODUCTIONS
Calc (. int x; .)
= "CALC" Expr<out x> (. System.out.println(x); .) .

Expr <out int x> (. int y; .)
= Term<out x>
{ '+' Term<out y> (. x = x + y; .)
}.

Term <out int x> (. int y; .)
= Factor<out x>
{ '*' Factor<out y> (. x = x * y; .)
}.

Factor <out int x>
= number (. x = Integer.parseInt(t.val); .)
| '(' Expr<out x> ')'.
END Calc.
```

Figura 12. Ejemplo de especificación COCO/R

Como se puede observar en la especificación las fases involucradas serán las tres, AL, AS y TDS. El objetivo principal de esta herramienta es permitir a los estudiantes ver el funcionamiento de un compilador como si se tratara del código de cualquier otro programa que se estuviera depurando, por ello el ámbito de uso de esta herramienta es educativo.

La visualización (véase la figura 13) se hace desde el punto de vista de depuración, por ello la visualización consiste en mostrar la línea de código actualmente en ejecución de tres elementos: programa principal (llamado compilador en VCOCO), AL y AS a esto se añaden dos vistas una que muestra la entrada a analizar y la otra que muestra las especificaciones léxica y sintáctica. Desde el punto de vista de la TDS, la concordancia entre la visualización y especificación es nula, ya que sólo se muestra a nivel léxico y sintáctico. Esta herramienta podría mostrar el código generado en la ventana del AS, puesto que las acciones semánticas no son sino código a ejecutar en determinados momentos del AS pero no queda claro si es capaz de mostrar una concordancia con la especificación en términos de qué acciones semánticas concretas se ejecutan o qué atributos manipulan y qué valores tienen estos. En la tabla 3 se puede observar que el resultado de la evaluación numérica para VCOCO es de 0 ya que no permite visualizar los atributos y los valores que tomen estos al momento de la ejecución del traductor, la creación del árbol sintáctico y la información de la acción semántica que se ejecuta en un momento determinado.

Tabla 3. Valoración numérica para VCOCO

Nivel de concordancia	VCOCO
AV	0
ArS	0
ASn	0
Total	**0**

Figura 13. Elementos visualizados con VCOCO [20]

En lo que se refiere a disponibilidad real de la herramienta, los autores no han podido encontrar una versión del software accesible para poder ejecutarla. Tampoco se ha encontrado documentación sobre evaluaciones educativas del uso de la herramienta.

3.4. EvDebugger

Es una plataforma para el desarrollo y la depuración de procesadores de lenguaje basados en la gramática de atributos orientada a especificaciones, cuyo principal objetivo es ayudar a los estudiantes a diseñar sus propios procesadores de lenguaje soportados por un depurador visual [21], por lo que el ámbito de uso de la herramienta es educativo.

EvDebugger se basa en tres componentes principales. En primer lugar, está el administrador de gramáticas (véase Fig. 14) que además es la pantalla inicial de la herramienta.

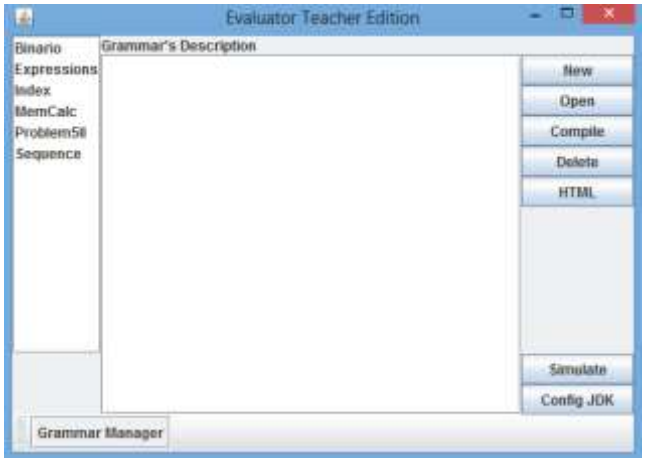

Figura 14. Administrador EvDebugger [21]

Por otro lado, se encuentra el editor de gramáticas (véase Fig. 15), que ofrece diferentes áreas especificando las partes de la especificación de la gramática de atributos: la definición de la gramática (véase Fig. 15a), la descripción de la gramática (véase Fig. 15b) y las clases semánticas (véase Fig. 15c). Finalmente, está el depurador (véase Fig. 16). Para acceder al depurador visual se debe compilar la gramática con éxito de modo que se pueda obtener un procesador de lenguaje correcto. La visualización se hace desde el punto de vista de depuración: (véase Fig. 16a) el árbol sintáctico, (véase Fig. 16b) los nodos de atributos, los atributos de la gramática (véase Fig. 16c), los valores semánticos de los atributos computados (véase Fig. 16d) y las herramientas del depurador (véase Fig. 16e). La concordancia entre la visualización y la especificación es total, puesto que visualiza producciones, acciones semánticas y atributos, como se puede ver en la figura 16. En la tabla 4, se puede observar que el puntaje de la valoración numérica obtenida por EvDebugger es de 1, permitiendo de esta manera visualizar los aspectos analizados para el estudio como son los nombres de los atributos y los valores que toman en el momento de la ejecución de traductor, la creación del árbol sintáctico y la información de las acciones semánticas que se ejecutan en un momento determinado.

Tabla 4. Valoración numérica para EvDebugger

Nivel de concordancia	Evdebugger
AV	0.3
ArS	0.3
ASn	0.4
Total	**1**

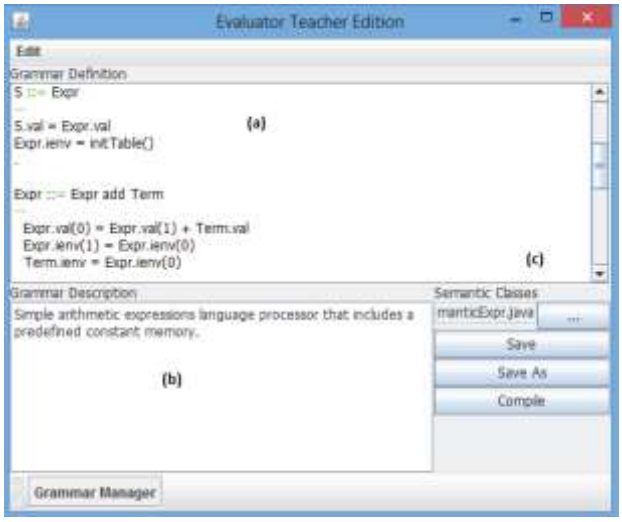

Figura 15. Editor de la gramática [21]

La especificación de la gramática debe escribirse con formato propio de la herramienta, para este caso, usa una notación similar a la que se utiliza en el curso de construcción de compiladores, en la (Fig. 15a) se puede observar un fragmento de dicho lenguaje. Las fases involucradas serán dos: AS y TDS.

En lo que se refiere a disponibilidad real de la herramienta, los autores no han podido encontrar una versión del software accesible para poder ejecutarla. En [21] se señala que se validó la herramienta durante el año académico 2013-2014 con los estudiantes de Compiladores de la Universidad do Minho del curso de Máster, realizando estudios que involucraron tanto a docentes como estudiantes. Las opiniones de los estudiantes fueron muy favorables. Desde el punto de vista de la eficacia educativa, se consiguió aumentar de forma significativa el porcentaje de aprobados en un examen realizado tras utilizar la herramienta, aunque el estudio fue con pocos estudiantes estos resultados son muy prometedores [22].

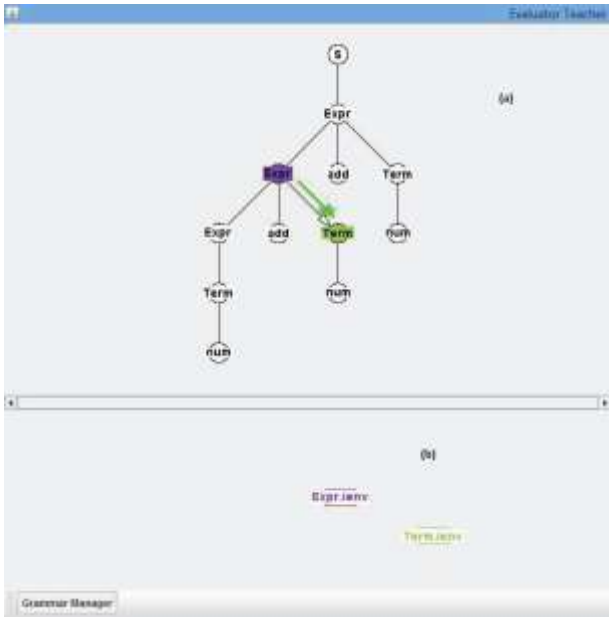

Figura 16. Herramientas del depurador visual [21]

4. Conclusiones

El contexto educativo de este trabajo es la enseñanza de compiladores y procesadores de lenguajes, más en concreto la fase de traducción dirigida por la sintaxis. Así, el objetivo principal de este trabajo es revisar las herramientas existentes que permiten visualizar la mencionada fase. De la búsqueda realizada, finalmente, se identificaron cuatro herramientas para su análisis detallado –LISA, CUPV, VCOCO y EvDebugger-, cuyos resultados exponemos a continuación.

El aspecto más destacable es la disponibilidad de las herramientas para su utilización. A pesar de que CUPV, VCOCO, EvDebugger son herramientas que se desarrollaron pensando en el ámbito educativo, no se pueden conseguir para poder utilizarlas en otros contextos diferentes al de su creación. En consecuencia, tampoco presentan las facilidades necesarias para su evaluación. LISA, que es una herramienta que se desarrolló pensando en el ámbito productivo ya que el propósito general, es la generación automática de compiladores a partir de las definiciones formales de lenguajes, sí estaba disponible y presentó facilidades para su evaluación. Sin embargo, la configuración de la plataforma tecnológica necesaria para utilizar LISA obliga a establecer unas condiciones que no corresponden con otros entornos actuales.

La principal ventaja de CUPV, LISA y EvDebugger frente a VCOCO es que muestran las fases del proceso de compilación de manera visual y en el caso de LISA interactiva, impulsando con esta característica la motivación que necesitarían los estudiantes de compiladores para interesarse más por la comprensión de los conceptos.

Es interesante destacar que EvDebugger utiliza un formato de especificación propio idéntico al utilizado en la asignatura en la que se usa y la concordancia entre esta especificación y la visualización es total. Aunque no hay una evidencia empírica clara EvDebugger logró obtener resultados positivos en las evaluaciones realizadas. Este ejemplo puede presentar un indicativo de lo que en principio parece lógico, que una buena concordancia así como coherencia con la asignatura puede resultar en evaluaciones positivas.

Como acabamos de ver, el formato propio de una especificación no es impedimento de buenos resultados, pero sí para la distribución de la herramienta fuera de su contexto de desarrollo. Por ello se cree positivo el uso de formatos estándar, como BNF o EBNF para el AS, y para la TDS las acciones semánticas incorporadas dentro de la gramática.

Para futuros trabajos que incorporen la construcción de una herramienta para la visualización de la TDS, se sugiere que: la herramienta está disponible para utilizarla fuera de su contexto de desarrollo, exista documentación sobre su evaluación, exista una interacción entre la especificación y la visualización, de modo que permita construir el árbol sintáctico con la asociación de los atributos con cada símbolo de la gramática, y además cada producción con el conjunto de acciones semánticas para ir calculando los valores de los atributos en cada nodo del árbol.

Agradecimientos

Este trabajo se ha financiado con el proyectoo de investigación TIN2015-66731-C2-1-R del Ministerio de Economía y Competitividad.

5. Referencias

[1] Ruiz, J. A., Martínez, M., & Sánchez, M. de la L. El impacto de las TICs en la calidad de la educación superior. *Journal of Research in Accounting and Management Science*, Vol. *1*, pp. *28 – 44*, 2016. Retrieved from http://ricca.umich.mx/index.php/ricca/article/view/1/10

[2] F. P. Guachún, "Aplicación e impacto de las TICs en la enseñanza de las matemáticas: una revisión sistemática," M.S. thesis, Fac. Filosofía, Letras y CC. de La Educación, Univ. de Cuenca, Ecuador, 2016.

[3] Gálvez Rojas, S., & Mora Mata, M. Á. (2005). Compiladores, traductores y compiladores con Lex/Yacc, JFlex/Cup y JavaCC. Universidad de Málaga.

[4] Aho, A. V, Lam, M. S., Sethi, R., & Ullman, J. D. (2008). Compiladores, principios, técnicas y herramientas. (Pearson Educación, Ed.) (second). México.

[5] Grune, D., Bal, H. E., Jacobs, C. J. H., & Langendoen, K. G. (2007). Diseño de compiladores modernos. (S. A. U. McGraw-Hill/Interamericana de España, Ed.) (primera). Madrid.

[6] D. E. Knuth, "Semantics of context-free languages", Mathematical systems theory, Vol. 2, No. 2, June, 1968, pp. 127-145. https://doi.org/10.1007/BF01692511

[7] R. M. Baecker "Sorting out sorting: A case study of software visualization for teaching computer science," in *Software Visualization: Programming as a multimedia experience*, Cambridge, MA: The MIT Press, 1998, ch *1*, pp. *369-381*

[8] J. Urquiza-Fuentes and J. Á. Velázquez-Iturbide, "A Survey of Successful Evaluations of Program Visualization and Algorithm Animation Systems," *Trans. Comput. Educ.* Vol *9*, No *2*, Article *9*, June, 2009, 10.1145/1538234.1538236

[9] S. H. Rodger, J. Genkins, I. McMahon, and P. Li. (2013, Julio). Increasing the experimentation of theoretical computer science with new features in JFLAP. Presented at ACM ITiCSE. [Online]. Available: http://dx.doi.org/10.1145/2462476.2466521

[10] J. Urquiza-Fuentes, F. J. Almeida-Martínez, A. Pérez-Carrasco y J. Ángel Velázquez-Iturbide, "Improving Students' Performance with Visualization of Error Recovery Strategies in Syntax Analysis", *Journal of Research and Practice in Information Technology*, Vol. *45*, No. *3/4*, August, 2013, pp. *237-250*

[11] Webster, J. y Watson, R. T. (2002). Analyzing the Past to Prepare for the Future: Writing a Literature Review. MIS Quarterly, 26(2), xiii–xxiii. Retrieved from http://www.jstor.org/stable/4132319

[12] F.J. Almeida-Martínez "Generación de Visualizaciones Educativas del Análisis Sintáctico," Ph.D. dissertation, ETSII, URJC, Móstoles, 2011.

[13] D. Rodríguez Cerezo, "Herramientas educativas para facilitar la adopción de la ingeniería de lenguajes software entre los desarrolladores informáticos," Ph.D. dissertation, DISIA, UCM, Madrid, 2016. [Online]. Available: http://eprints.ucm.es/42890/

[14] J. Urquiza-Fuentes, F. Manso, J. Á. Velázquez-Iturbide y M. Rubio-Sánchez. (2011, Julio). Improving compilers education through symbol tables animations. Presented at ACM ITiCSE'11. [Online]. Available: https://doi.org/10.1145/1999747.1999805

[15] Mernik, M., Lenic, M., Enis, A., & Zumer, V. (2000). Compiler / Interpreter Generator System LISA. *Proceedings of the 33rd Annual Hawaii International Conference on IEEE*, *0*(c), 1–10. Retrieved from https://pdfs.semanticscholar.org/77d4/872dbeff9c289b5989fc483488fee981d6a5.pdf

[16] Rebernak, D., Mernik, M., & Jo, M. (2000). AspectLISA : an aspect-oriented compiler construction system based on attribute grammars, 1–18.

[17] Mernik, M., & Žumer, V. (2005). Incremental programming language development. Computer Languages, Systems & Structures, 31(1), 1–16. https://doi.org/10.1016/j.cl.2004.02.001

[18] Cruz, D., & Henriques, P. (2002). A little manual for LISA Universidade do Minho. Retrieved from http://www4.di.uminho.pt/~gepl/LP/manuals/lisaManual.pdf

[19] Kaplan, A., & Shoup, D. (2000). CUPV - A Visualization Tool for Generated Parsers. Austin, Texas, USA: SIGCSE '00 Proceedings of the thirty-first SIGCSE technical symposium on Computer science education. https://doi.org/10.1145/330908.331801

[20] R. D. Resler and D. M. Deaver (1998, Julio). VCOCO: a visualisation tool for teaching compilers. Presented at ACM ITiCSE'98. [Online]. Available: http://dx.doi.org/10.1145/282991.283123

[21] Rodríguez-Cerezo, D., Henriques, P., & Sierra, J.-L. (2014). Atribute grammars made easier: EvDebugger (pp. 23–28). Logroño, Spain: Computers in Education (SIIE), 2014 International Symposium on, IEEE. https://doi.org/10.1109/SIIE.2014.7017699

[22] D. Rodriguez-Cerezo, M. Gómez-Albarrán, and J.-L. Sierra-Rodríguez. (2013, Julio). Interactive educational simulations for promoting the comprehension of basic compiler construction concepts. Presented at ACM ITiCSE. [Online]. Available: http://dx.doi.org/10.1145/2462476.2462498521