Actas de las Jenui, vol. 6. 2021.
Páginas: 171-178

Hacia un modelo genérico de visualización de traductores dirigidos por la sintaxis

Ángel F. Sánchez-Granados, Jaime Urquiza-Fuentes,
Adrián García-Oller & José M. Loeches-Ruiz
LITE - Laboratorio de Tecnologías de la Información en la Educación
Universidad Rey Juan Carlos
28933 Móstoles (Madrid)
{angelfrancisco.sanchez,jaime.urquiza}@urjc.es,
{a.garciaoll,jm.loeches}@alumnos.urjc.es

Resumen

Este trabajo presenta una herramienta de visualización que pretende ayudar a comprender los conceptos de los traductores dirigidos por la sintaxis, proporcionando una representación gráfica de conceptos tan abstractos. La herramienta que se describe consta de una interfaz de visualización y una API. La interfaz de visualización permite a profesores y estudiantes ver una representación gráfica del proceso de ejecución del traductor dirigido por la sintaxis observando detalles como las acciones semánticas ejecutadas o los valores delos atributos de los símbolos. La API permite al usuario anotar una especificación del traductor dirigido por la sintaxis de forma que durante su ejecución genere una traza que permita su posterior visualización con la interfaz. Esta división entre interfaz y API busca que las visualizaciones puedan ser independientes del software utilizado para la generación del traductor, tratando así de sortear alguna de las limitaciones que tienen otras herramientas existentes.

Abstract

In this work, we present a visualization tool aimed at supporting students' comprehension of syntax directed translators concepts by providing graphical representations of such abstract concepts. The tool is made up of two parts: a visualization interface and an API. The visualization interface provide teachers and students with a graphical representation of the execution process of the syntax directed translator. This representation provide a detailed view of the executed semantic actions and the values of the symbol attributes. The API allows the user to annotate the specification of the syntax directed translator. Thus, its execution generates a trace that will be used by the interface to produce the visualization. This division between the interface and the API supports the independence between the visualizations and the parser generator tool, avoiding some of the limitations identified in other existing tools.

Palabras clave

Visualización del software, procesadores de lenguajes traductores dirigidos por la sintaxis, enseñanza de la informática.

1. Introducción

La disciplina de la visualización del software aplicada al entorno educativo tiene un largo recorrido, sirva como ejemplo las animaciones de algoritmos de ordenación realizadas por Ronald Baecker en los años 60 [3]. Una de las posibles aplicaciones que podemos encontrar de esta especialidad es la visualización de conceptos abstractos, que nos permitirá conseguir que los estudiantes sean capaces de formar un mejor modelo mental del concepto a estudiar. Por otro lado, tanto Naps et al. [9] como Hundhausen et al. [5] concluyen que además de la visualización *per se* es determinante la forma en que esta se usa por parte de los estudiantes.

En la enseñanza de la Informática, la asignatura de Procesadores de Lenguajes resulta habitualmente compleja para los estudiantes debido en parte a la dificultad de aplicar a situaciones reales los conceptos abstractos que se trabajan [4]. Una de las ramas fundamentales que la componen es la traducción dirigida por la sintaxis (TDS), que se basa en conceptos de analizadores sintácticos enriquecidos con la posibilidad de producir información y transmitirla a través del árbol sintáctico enriquecido para ejecutar un lenguaje de programación [7], y que resulta de gran importancia a la par que complejo. La TDS proporciona, por tanto, la base teórica para entender algunas etapas elementales de los com-

piladores, como pueden ser la generación de código intermedio o la comprobación de tipos. Precisamente, la visualización pretende servir de enlace entre los conceptos teóricos y su aplicación práctica.

Este trabajo se centrará en la aplicación de la visualización del software a la TDS con el objetivo de que los estudiantes mejoren su comprensión. El resto del artículo se estructura como sigue. La sección 2 describe los trabajos relacionados sobre la visualización de Procesadores de Lenguajes. La herramienta se describe en la sección 3 y en la sección 4 se muestra un ejemplo de uso. La sección 5 menciona la evaluación preliminar de la herramienta. Finalmente, la sección 6 expone las conclusiones y los trabajos futuros.

2. Trabajos relacionados

El desarrollo de traductores dirigidos por la sintaxis (en adelante traductores) es una tarea compleja y por ello existen una gran variedad de herramientas diseñadas para apoyarla. Algunas se centran exclusivamente en el desarrollo, por ejemplo CUP[1] o ANTLR[2], aunque también se usan en entornos educativos. El esquema de funcionamiento básico consiste en que el desarrollador produce una especificación del traductor, las herramientas antes mencionadas se usan para generar el código fuente del traductor a partir de la especificación. Una vez generado dicho traductor, este se ejecutará como cualquier otra aplicación procesando la cadena de entrada y realizando las tareas incluidas en la especificación.

Por otra parte, también existe software relacionado donde la visualización asume un rol determinante, como los sistemas VAST, LISA, CUPV, VCOCO y EvDebugger.

VAST [2] es una herramienta creada específicamente para el entorno educativo que permite generar y manipular árboles sintácticos independientemente del analizador con el que ha sido creado, ascendente (CUP) o descendente (ANTLR). VAST procesa la especificación sintáctica y anota de forma automática dicha especificación de forma que cuando termina de procesar una cadena genera la visualización. La interfaz de visualización permite observar la pila de ejecución, el árbol sintáctico e incluso la recuperación de errores sintácticos [13]. Aunque es un entorno con claro objetivo educativo, solo se centra en el análisis sintáctico.

LISA [8] es una herramienta de generación de intérpretes y compiladores que dispone de una especificación gramatical propia. La versión 2.2 soporta analizadores de tipo LL, LR y LALR. Su planteamiento se centra en el desarrollo de lenguajes de dominio específico[3], por lo que su enfoque no es educativo. La especificación del traductor se debe realizar con una notación propia de la herrameinta. Su interfaz gráfica permite visualizar el árbol sintáctico, el grafo de dependencias y los valores de los atributos del árbol enriquecido. A pesar de tener un enfoque profesional y no haber sido evaluada formalmente, existen experiencias donde los estudiantes reflejan una opinión positiva de la herramienta además de potenciar su participación activa en las clases [10].

CUPV [6] es una herramienta de visualización para las gramáticas CUP, de modo que aprovecha su especificación para la visualización, y está orientada específicamente para el entorno educativo. Permite visualizar atributos del proceso de compilación tales como valores semánticos, reducciones, el árbol sintáctico o la entrada. Actualmente no se encuentra disponible para su descarga.

VCOCO [11] es un software que permite visualizar la ejecución de traductores dirigidos por la sintaxis generados con la herramienta COCO/R[4]. Por ello, utiliza su especificación léxica, sintáctica y de TDS. Esta herramienta permite visualizar la información léxica y sintáctica relativa a la instrucción de código que está en ejecución. Su interfaz de visualización usa el enfoque de un depurador. Esta herramienta tampoco se encuentra disponible para su descarga.

EvDebugger [12] es una herramienta de corte educativo que permite diseñar traductores con la ayuda de un depurador visual. Con una especificación gramatical propia, a través de su depurador se puede visualizar el árbol sintáctico con sus nodos de atributo, los atributos de la gramática y los valores semánticos de los atributos computados. Su evaluación recogió buenas opiniones de estudiantes y profesores pero no se ha podido encontrar enlace de descarga para esta herramienta.

Es obvio que ha habido un interés por utilizar las tecnologías de visualización con los traductores. De las herramientas que hemos encontrado, las que más se acercan a nuestro propósito son LISA y EvDebugger, aunque existen ciertas lagunas por cubrir. Ambas utilizan un lenguaje de especificación propietario, la primera no está diseñada para el ámbito educativo aunque se haya hecho una evaluación referente al analizador sintáctico y la segunda no se puede descargar para su utilización.

Nuestro objetivo se centra en conseguir una herramienta que pueda visualizar los detalles de la traducción dirigida por la sintaxis y que sea tan independiente como sea posible de las herrameintas de desarrollo de traductores, de forma que no haya que aprender un

[1] `http://www2.cs.tum.edu/projects/cup/`, 2021
[2] `https://www.antlr.org/`, 2021
[3] DSL, Domain Specific Languages
[4] http://www.ssw.uni-linz.ac.at/Coco/, 2021

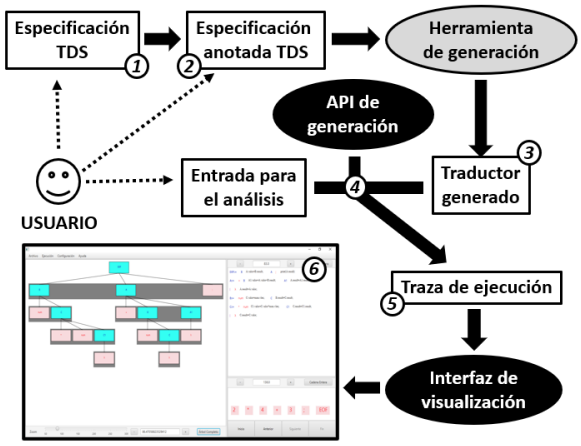

Figura 1: Arquitectura del sistema.

nuevo lenguaje de especificación ni ceñirse a un único tipo de generador de traductores.

3. Descripción de la herramienta

El objetivo principal de esta herramienta es visualizar el funcionamiento de un traductor. Aun así, también se quiere conseguir que esta visualización sea independiente de las herramientas de desarrollo de traductores.

La fig. 1 muestra un esquema del uso y la arquitectura del sistema. Este se compone de una interfaz de visualización y una API de generación de información a visualizar. La utilización sería de la siguiente forma. En primer lugar, el usuario escribe su especificación del traductor en cualquiera de los lenguajes de generación soportados por la herramienta (1), y a continuación se anota con llamadas a una API dando lugar a la *especificación anotada* (2). Se genera el traductor utilizando la especificación anotada (3). Cuando este se ejecuta (4) con una cadena de entrada, también proporcionada por el usuario, produce como efecto lateral –de las llamadas a la API– una traza de ejecución (5) que será la información utilizada por la interfaz para mostrar la visualización de la ejecución del traductor (6).

Por lo tanto, el sistema se basa en la visualización post-mortem, mostrando la ejecución del traductor una vez esta haya terminado. A continuación, se describe el diseño tanto de la interfaz como de la API.

3.1. Interfaz de visualización

La ejecución de un TDS depende del modelo de evaluación de las acciones semánticas asociadas a las producciones de la gramática. Nuestro sistema utiliza la evaluación en tiempo de compilación [1], así, las acciones semánticas que se evalúan son aquellas correspondientes a las producciones que el analizador sintáctico aplica en cada momento en función de la cadena de entrada que se esté analizando. Por ello, se necesita tener en cuenta el tipo de analizador que se está utilizando y así visualizar correctamente el proceso de construcción del árbol sintáctico. La visualización de la construcción del árbol lleva consigo la visualización de los diferentes símbolos gramaticales. Además, para visulizar la evaluación de las acciones semánticas (la ejecución del traductor) se mostrará la versión enriquecida del árbol sintáctico, proporcionando el valor de los atributos de los símbolos gramaticales así como dando información sobre las acciones semánticas ejecutadas que manipulan y modifican el valor de dichos atributos.

Todo ello nos lleva a identificar tres elementos principales en nuestra interfaz de visualización: la especificación del traductor, la entrada con que se ejecutará el traductor y el estado del árbol sintáctico enriquecido en cada momento. La especificación del traductor es necesaria por varias razones. En primer lugar, el objetivo de la enseñanza de la TDS es que los estudiantes sean capaces de crear especificaciones de traductores que funcionen cumpliendo los requisitos que se les planteen. En segundo lugar, y como consecuencia de la anterior, para poder crear especificaciones, los estudiantes deberán entender cómo se ejecutan estas, por lo que debe aparecer simultáneamente con el resto de elementos que dan información sobre su ejecución. Así, el siguiente elemento será la cadena de entrada, puesto que es quien guía la ejecución del traductor y por lo tanto proporcionará la información sobre los distintos instantes de ejecución del mismo. Finalmente, el árbol sintáctico enriquecido es propiamente la visualización del estado interno del traductor, que se asemeja al modelo mental utilizado generalmente en la enseñanza de procesadores de lenguajes [1]. Siendo estos tres elementos los fundamentales para la compresión de la ejecución de un traductor, su visualización debe realizarse de forma simultánea, como se ha dicho anteriormente, pero además sincronizada.

La fig. 2 muestra una captura de la interfaz de visualización, donde la parte izquierda corresponde a la visualización del árbol sintáctico enriquecido, la parte superior derecha corresponde a la especificación del traductor y la parte inferior derecha corresponde a la visualización del estado de la cadena de entrada.

En la especificación del traductor se muestran los dos elementos principales de la misma: las reglas sintácticas y las acciones semánticas. Como el objetivo es centrarse en visualizar la ejecución del traductor, hemos decidido esconder los detalles específicos asociados al generador de TDS concreto. Además, el usuario puede elegir ocultar o mostrar las acciones semánti-

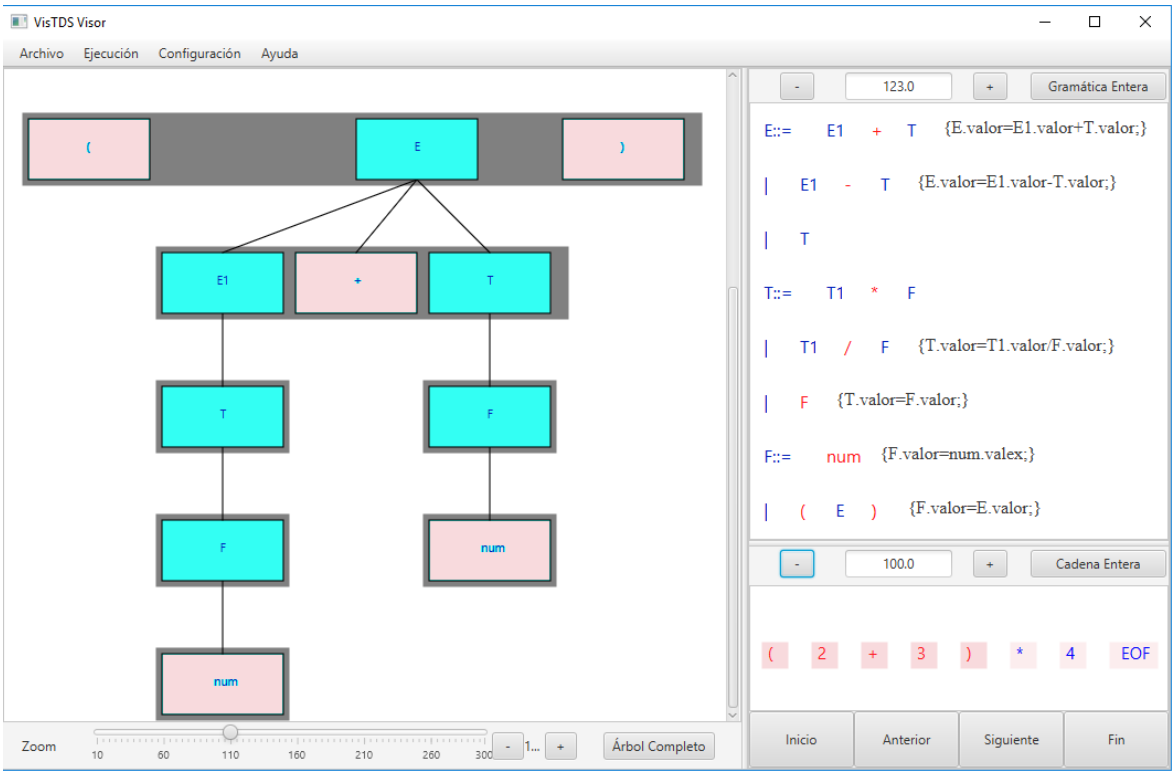

Figura 2: Captura de pantalla de la interfaz de visualización: árbol sintáctico enriquecido (zona izquierda), especificación del TDS (zona superior derecha) y estado de la cadena de entrada (zona inferior derecha).

cas asociadas a determinadas reglas. Por ejemplo, en la fig. 2 se ocultan las acciones semánticas de las producciones `E ::= T` y `T ::= T1 * F`. De esta forma se consigue una visualización concisa de la especificación adaptada a las preferencias del usuario.

En la cadena de entrada se muestran literalmente los símbolos terminales (lexemas de los tokens) que la forman. Estos símbolos tendrán una representación diferente según hayan sido procesados o estén pendientes de procesar. Así, en la fig. 2 se ve que se han procesado símbolos de la cadena de entrada hasta el quinto token, el resto no se han procesado todavía. Finalmente, como se ha mencionado, la cadena de entrada guía la ejecución del traductor según la va procesando, por esa razón se permite navegar por la ejecución del traductor utilizando la cadena de entrada. Así, el usuario puede seleccionar un token concreto y la visualización del árbol sintáctico enriquecido que se mostrará corresponderá al estadeo ejecución cuando el traductor haya llegado a procesar dicho token.

La visualización del árbol sintáctico enriquecido comprende dos partes: el árbol sintáctico y la información de valores de atributos que muestra el resultado de las acciones semánticas. El árbol sintáctico se visualiza según su proceso de construcción –ascendente, como en la fig. 2, o descendente–, mostrando de forma diferenciada los símbolos terminales, los no terminales y su asociación mediante producciones sintácticas. Por ejemplo, la fig. 2 muestra la aplicación de las producciones (reducciones, por ser de tipo ascendente):

`F ::= num`, `T ::= F`, `E ::= T` y `E ::= E1 + T`

La versión enriquecida del árbol sintáctico proporciona información bajo demanda sobre los valores de los diferentes atributos asociados a cada símbolo, la acción semántica ejecutada e incluso las explicaciones que el profesor quiera incluir. En la fig. 3 se pueden ver ejemplos de estos tres casos.

Lo que se acaba de describir es la información que ofrece de forma estática la visualización de un estado de ejecución concreto. Pero la herrameinta también permite ver la evolución de este estado según se va procesando la cadena de entrada, ofreciendo por tanto una visualización dinámica de la ejecución del traductor.

Además de la visualización de los tres elementos anteriores, la interfaz tiene otras características, más básicas pero muy útiles. Así, permite configurar la visualización los elementos anteriores, asegurando una coherencia en la representación de símbolos terminales y no terminales en todas las vistas. También permite hacer zoom del árbol enriquecido, de forma que en los casos en que dicho árbol se haga demasiado grande se pueda seguir teniendo una vista global del mismo. Y finalmente, también se permite la navegación paso a paso mediante teclado y ratón.

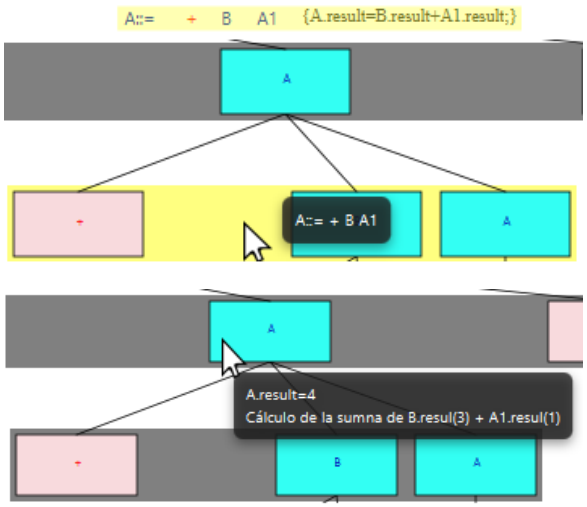

Figura 3: Ejemplos de información mostrada por el árbol sintáctico enriquecido. Arriba, producción y destacado de la acción semántica en la especificación. Abajo, valor del atributo y explicaciones añadidas.

Por último, mencionar que el diseño de la interfaz comenzó con un proceso de prototipado, al que siguió su implementación utilizando Java Swing[5] y la API JGraph [6] para la representación del árbol. Finalmente, la interfaz evolucionó utilizando JavaFX[7].

El diseño de la interfaz ha seguido un proceso de prototipado con diseño participativo contando con el profesor de la asignatura de Procesadores de Lenguajes y cuatro estudiantes con conocimientos contrastados de la misma.

3.2. API de generación

El objetivo de la API es generar la información de la ejecución del traductor, la traza de ejecución, que será interpretada y visualizada por la interfaz antes descrita. La traza de ejecución se generará en formato XML, como efecto lateral de la ejecución del traductor anotado con llamadas a la API generadora. El proceso de anotación consiste principalmente en añadir llamadas a los métodos de la API dentro de las acciones semánticas de la especificación del traductor. El diseño de la API persigue principalmente dos objetivos, por un lado la facilidad de uso y por otro la cobertura de los dos tipos de analizadores: ascendentes y descendentes. Para ello, hemos trabajado con dos herramientas ampliamente utilizadas y mencionadas anteriormente: CUP para los analizadores ascendentes y ANTLR para los descendentes. Aunque los

[5]Package javax.swing, `https://docs.oracle.com/javase/7/docs/api/javax/swing/package-summary.html`, 2021

[6]JGraph, `https://github.com/jgraph`, 2021

[7]JavaFX, `https://openjfx.io/`, 2021

```
A ::= plus B:b A:a1 {: RESULT=b + a1; :}
|{: RESULT=0; :};
```

```
A ::= plus B:b A:a1 {:
    writer.addStepNonTerminal("A", "result", b + a1,
        "A::= + B A1 {A.result=B.result+A1.result;}");
    RESULT=b + a1;
:}

|{:
    writer.addStepLambda("A", "result", 0, "{A.result=0;}");
    RESULT=0;
:};
```

Figura 4: Ejemplo de la anotación de una especificación CUP. Arriba, fragmento de una especificación CUP sin anotaciones. Abajo, fragmento de una especificación CUP anotada.

métodos utilizados en ambos tipos de analizadores son similares, existen pequeñas diferencias debido por ejemplo a que CUP solo permite atributos sintetizados mientras que ANTLR permite utilizar tanto atributos sintetizados como heredados. Una parte de la anotación consiste en la creación de clases para los símbolos gramaticales (que ayudan a manipular la información de la traza correspondiente cada símbolo) y su inclusión en la especificación. Al ser totálmente mecánico, este proceso se ha automatizado mediante el comando:

```
java -jar VisTDSApiXMLCreator.jar
<especificacion>
```

A continuación se describen las anotaciones para los analizadores ascendentes (con CUP) y descendentes (con ANTLR). Independientemente del tipo de analizador y antes anotar manualmente la especificación, habrá que ejecutar el comando anteriormente mencionado que ejecutará las acciones necesarias según la especificación (CUP o ANTLR).

Anotando especificaciones CUP

Tras ejecutar el comando `VisTDSApiXMLCreator`, además de crear las clases asociadas a cada símbolo, se añadirá a la definición de los símbolos terminales su especificación como `String`, lo que permitirá su visualización en la interfaz. Una vez hecho esto basta con añadir las siguientes llamadas en las producciones de la gramática. Si la producción no corresponde a una regla borradora se añade la siguiente llamada:

```
writer.addStepNonTerminal(
    antecedente, nombre-atr-sintetizado,
    valor-atr, texto);
```

Si la regla produce lambda, el método usado es `writer.addStepLambda` con los mismos argumentos. Un ejemplo del uso de estos métodos se puede ver en la fig. 4.

```
exp returns [Exp expO]:
    bO=b
    aO=a[Integer.parseInt(((ExpContext)_localctx).bO.bO.getValue())]
    ';'
    {
        _localctx.expO=expO;
    }
;
```

```
exp returns [Exp expO]:
   {
       Node node=writer.addStepNonTerminal("EXP", null, null, "null");
   }
   bO=b
   aO=a[Integer.parseInt(((ExpContext)_localctx).bO.bO.getValue())]
   ';'
   {
       writer.updateNonTerminals(
           "EXP::= B {A.valor=B.result;} A ; {print(A.result);}",
           ((ExpContext)_localctx).aO.aO.getValue(),
           ((ExpContext)_localctx).bO.bO);
       _localctx.expO=expO;
       writer.writeXML();
   }
;
```

Figura 5: Ejemplo de la anotación de una especificación ANTLR. Arriba, fragmento de una especificación ANTLR sin anotaciones. Abajo, fragmento de una especificación ANTLR anotada.

Anotando especificaciones ANTLR

Al ser un analizador descendente, el árbol se crea utilizando un recorrido en profundidad. Por ello necesitamos realizar dos anotaciones, una al comienzo y otra al final de la regla. La primera se realiza con la siguiente llamada:

```
writer.addStepNonTerminal(
    antecedente, nombre-atr-heredado,
    nombre-atr-sintetizado,
    referencia-atr-heredado);
```

La llamada al final de la regla sigue este formato:

```
writer.updateNonTerminals(
    regla-gramatical,
    referencia-atr-sintetizado,
    referencia-primer-simbolo);
```

Al igual que en el caso de CUP, las reglas lambda también tienen una anotación especial:

```
writer.addStepLambda(
    antecedente, nombre-atr-heredado,
    nombre-atr-sintetizado,
    referencia-atr-heredado, regla-gramatical);
```

En la fig. 5 se muestra un ejemplo de anotación.

4. Ejemplo de uso

El objetivo principal de esta herramienta es que los estudiantes puedan entender el funcionamiento de la TDS mostrando tanto los valores de los atributos como la relación entre la aplicación de las reglas sintácticas y la ejecución de acciones semánticas.

Existen dos notaciones para especificar traductores: definiciones dirigadas por sintaxis (DDS) y esquemas de traducción (EDT). Las primeras son más simples, ya que se suelen asociar a un funcionamiento ascendente (usando frecuentemente recursividad por la izquierda) ejecutando acciones semánticas al final de las producciones y trabajando frecuentemente con atributos sintetizados. Los segundos son más complicados, se suelen asociar a un funcionamiento descendente (usando recursividad por la derecha), permitiendo ejecutar acciones en cualquier punto del consecuente y haciendo uso tanto de atributos heredados como sintetizados.

Un problema típico de esta asignatura es que los estudiantes tratan de aplicar las soluciones más sencillas con DDS de forma frecuente, incluso cuando lo que deben usar son EDT. El problema de estos fallos es que son difíciles de detectar si no se ejecuta el traductor.

Se puede usar como ejemplo la típica gramática de operaciones aritméticas. Para los analizadores ascendentes, esta gramática es ideal, ya que permite implementar de forma inherente la asociatividad a izquierdas gracias a la recursividad a izquierdas de las reglas. El traductor es muy fácil de entender puesto que para cada regla se asocia una acción semántica que ejecuta la operación usando los operandos –atributos de los símbolos del consecuente– y devuelve el resultado en el atributo sintetizado del antecedente.

```
S ::= E {print(E.v);}
E ::= E1 + cte {E.v = E1.v + cte.v;}
E ::= E1 - cte {E.v = E1.v - cte.v;}
E ::= cte {E.v = cte.v;}
```

El problema aparece con las gramáticas recursivas a derechas, donde la asociatividad no se implementa con la recursividad de la gramática y sería necesario usar un EDT. Un ejemplo de esta podría ser:

```
E ::= cte Ep
Ep ::= + cte Ep1 | - cte Ep1 | λ
```

A pesar de no ser recursiva por la izquierda, los estudiantes tratan de implementar el mismo esquema que en los traductores ascedentes: acciones semánticas al final de las producciones y uso de atributos sintetizados. Para ello, en las producciones con operación, ejecutan esta usando como operandos los símbolos del consecuente –la constante `cte` y el atributo sintetizado del símbolo recursivo `Ep`– viéndose obligados a asignar un valor inicial neutro en la regla borradora de `Ep`, en el caso de la suma es el `0`. Así, para la suma, el traductor quedaría:

```
Ep ::= + cte Ep1 {Ep.v = cte.v + Ep1.v;}
Ep ::= λ {Ep.v = 0;}
```

Como la operación aritmética de la suma tiene la propiedad asociativa, al ejecutar las sumas con asociatividad por la derecha, el traductor da resultado correcto. Y los estudiantes no se dan cuenta de que han cambiado la asociatividad.

Los estudiantes usan el mismo esquema de acciones

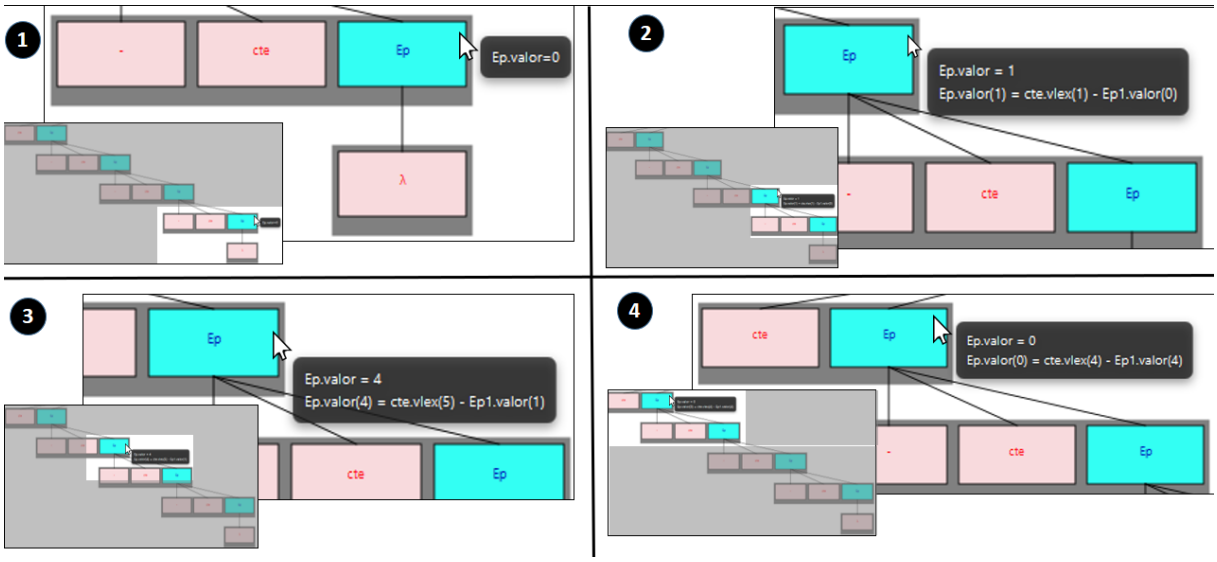

Figura 6: Capturas de la visualización de los resultados de cada operación procesada (ordenadas del 1 al 4) para la cadena `15-4-5-1`. Cada paso cuenta con una vista global del árbol que identifica a la operación ejecutada.

semánticas y atributos sintetizados para la regla de la resta. Como esta operación aritmética no tiene la propiedad asociativa, el resultado es erróneo. El problema es que frecuentemente, los estudiantes no identifican el cambio de asociatividad provocado por el cambio de recursividad. Con esta herramienta, los estudiantes pueden ver paso a paso los cálculos que se van realizando, viendo el orden en que se ejecutan las acciones. En la fig. 6 se puede cómo la herramienta permite a los estudiantes ver los cálculos parciales de las restas al procesar la cadena `15-4-5-1`, mostrando por qué su traductor da como resultado `15` en vez de `5`.

5. Evaluación de la herramienta

El desarrollo de la herramienta finalizó en el primer semestre del curso 2019-2020, y su evaluación estaba planificada para el segundo semestre. Debido a la situación actual, bajo las restricciones impuestas por la pandemia COVID-19, no se ha podido plantear un experimento controlado que permita realizar la mencionada evaluación, siendo postpuesto al segundo semestre del curso 2020-2021.

Aún así, se ha contado con las impresiones recogidas de estudiantes con conocimientos contrastados sobre la asignatura. Esta recolección de información se ha realizado durante el proceso de diseño y desarrollo de la interfaz. Los estudiantes se mostraron entusiasmados con la posibilidad de ver una representación de la ejecución del TDS. También hicieron multitud de comentarios que han servido para mejorar diversos detalles de la interfaz. Es importante destacar que, al ser un tema muy complejo de la asignatura, el nivel de exigencia para con la herramienta por parte de los estudiantes es muy alto.

6. Conclusiones y trabajo futuro

En este trabajo se describe una herramienta que permite visualizar la ejecución de traductores dirigidos por la sintaxis partiendo de sus especificaciones, creadas por los propios usuarios (tanto profesores como alumnos) y con sus propias cadenas de entrada. Resultados preliminares en campos relacionados [5, 13, 10, 12] nos indican que el uso de la visualización de software en la enseñanza de la traducción dirigida por la sintaxis puede ayudar a su aprendizaje.

Tras analizar las herramientas relacionadas existentes se ha podido comprobar que ninguna es capaz de trabajar con diferentes notaciones de especificaciones, algunas se restringen a un determinado tipo de analizador sintáctico [6, 11] mientras que otras tienen limitaciones de interacción y/o despliegue [8, 12], barreras que pretenden eliminarse con esta propuesta.

Con esta herramienta los usuarios pueden anotar de forma transparente sus propias especificaciones recibiendo feedback sobre su funcionamiento, sin necesidad de disponer de una batería de ejemplos previos. A nivel de desarrollo, se ha concebido para ser flexible tanto para la adaptación de nuevas notaciones de especificaciones como para ser independiente de la interfaz de visualización, teniendo la capacidad de generar la especificación de salida adaptada a las necesidades del visualizador utilizado. Gracias a la API, los textos añadidos a la aplicación de producciones y valores de atributos permiten a los profesores variar el nivel de las

explicaciones proporcionadas desde un mínimo nivel a modo de depurardor hasta un nivel de explicación textual detallada. Estas animaciones se pueden utilizar tanto en clase como en el tiempo de estudio propio, algo muy necesario en el contexto educativo actual de restricciones impuestas por la COVID-19. Como el progreso de la animación está totalmente controlada por el usuario, el profesor podría hacer preguntas del estilo *¿qué pasará a continuación?* o *¿cuál será el valor de este atributo?* Por otro lado, los estudiantes pueden reproducir tantas veces como deseen la animación así como centrarse únicamente en los puntos que les interesa de la ejecución del traductor dirigido por la sintaxis.

La versión actual de la herramienta carece de una evaluación formal empírica. Sin embargo, se ha contado con las impresiones de diez estudiantes con conocimientos contrastados de la asignatura, cuatro de ellos durante el diseño y otros seis que llevaron a cabo una evaluación heurística. Además de ayudar a detectar mejoras, mostraron una opinión muy positiva sobre dos aspectos: la posibilidad de *ver* el comportamiento del traductor y la interactividad ofrecida por la herramienta.

En cuanto a los trabajos futuros, el más inmediato es la anotación automática de las especificaciones. Sin embargo, antes realizaremos una evaluación empírica de la interfaz. Por otro lado, la API debe permitir la adaptación a cualquier herramienta de generación de TDSs. Finalmente, se afrontará la liberación del código fuente para que esté a disposición de la comunidad de desarrolladores, así como el desarrollo de una versión cloud de esta herramienta que pueda ser utilizada en cualquier lugar y desde cualquier dispositivo sin necesidad de instalar ningún tipo de software.

Agradecimientos

Este trabajo ha sido financiado por el Ministerio de Economía y Competitividad (ref. TIN2015-66731-C2-1-R) y la Comunidad de Madrid, con el proyecto e-Madrid-CM (P2018/TCS-4307) , también cofinanciado por los fondos estructurales (FSE y FEDER).

Referencias

[1] Alfred V. Aho, Monica S. Lam, Ravi Sethi y Jeffrey D. Ullman. *Compilers: Principles, Techniques, and Tools (2nd Edition)*. Addison-Wesley, USA, 2006.

[2] Francisco J. Almeida-Martínez, Jaime Urquiza-Fuentes, y Ángel Velázquez-Iturbide. Visualization of syntax trees for language processing courses. *J. Univers. Comput Sci.*, 15(7):1546–1561, apr 2009.

[3] Ronald M. Baecker. *Interactive Computer-Mediated Animation*. PhD thesis, M.I.T. Department of Electrical Engineering, April 1969.

[4] Michael Hewner. Undergraduate conceptions of the field of computer science. In *ACM ICER 2013*, pages 107–114, New York, NY, USA, 2013. ACM.

[5] Christopher D. Hundhausen, Sarah A. Douglas y John T. Stasko. A meta-study of algorithm visualization effectiveness. *J. Visual. Lang. Comput.*, 13(3):259–290, 2002.

[6] Alan Kaplan y Denise Shoup. Cupv—a visualization tool for generated parsers. *SIGCSE Bull.*, 32(1):11–15, March 2000.

[7] Donald E. Knuth. Semantics of context-free languages. *Math. Syst. Theory*, 2:127–145, 06 1968.

[8] Marjan Mernik y Viljem Zumer. An educational tool for teaching compiler construction. *IEEE Trans. Educ.*, 46(1):61–68, 2003.

[9] Thomas L. Naps, Güido Rößling, Vicky Almstrum, Wanda Dann, Rudolph Fleischer, Christopher D. Hundhausen, Ari Korhonen, Lauri Malmi, Myles McNally, Susan Rodger y Ángel Velázquez-Iturbide. Exploring the role of visualization and engagement in computer science education. *SIGCSE Bull.*, 35(2):131–152, June 2002.

[10] Damian Nicolalde-Rodríguez y Jaime Urquiza-Fuentes. Educational impact of syntax directed translation visualization, a preliminary study. In *IEEE VL/HCC 2018*, pages 313–314, 2018.

[11] Daniel Resler y Deam M. Deaver. Vcoco: A visualisation tool for teaching compilers. *SIGCSE Bull.*, 30(3):199—-202, August 1998.

[12] Daniel Rodríguez-Cerezo, Pedro R. Henriques, y José-Luis Sierra. Attribute grammars made easier: Evdebugger a visual debugger for attribute grammars. In *SIIE 2014*, pages 23–28, 2014.

[13] Jaime Urquiza-Fuentes, Francisco J. Almeida-Martínez, y Ángel Velázquez-Iturbide. Improving students' performance with visualization of error recovery strategies in syntax analysis. *J. Res. Pract. Inf. Technol.*, 45(3/4):237–250, Aug 2013.