Actas de las Jenui, vol. 6. 2021.
Páginas: 163-170

Generación automática de preguntas cloze para cuestionarios Moodle sobre análisis léxico

Roberto Izquierdo Amo
Media & Entertainment Innovation.
Telefónica I+D
Rda. Comunicación, s/n, 28050, Madrid
roberto.izquierdoamo@telefonica.com

César I. García Osorio
Depto. de Ing. Informática.
Universidad de Burgos
Avda. Cantabria, s/n, 09006, Burgos
cgosorio@ubu.es

Pedro Latorre Carmona
Depto. de Ing. Informática.
Universidad de Burgos
Avda. Cantabria, s/n, 09006, Burgos
plcarmona@ubu.es

José A. Barbero Aparicio
Depto. de Ing. Informática.
Universidad de Burgos
Avda. Cantabria, S/N, 09006, Burgos
jabarbero@ubu.es

Resumen

Los cuestionarios de Moodle son una manera conveniente de afrontar la evaluación online, tan relevante en época de pandemia. Desafortunadamente, su preparación es lenta, engorrosa y propensa a fallos. Además de diseñar la pregunta, luego hay que crearla en Moodle. Si tiene imágenes, hay que crearlas y subirlas a Moodle. También hay que configurar la calificación asociada a la respuesta correcta y a las opciones alternativas incorrectas. Si este proceso se tiene que repetir con muchas preguntas, el tiempo necesario termina siendo prohibitivo. En este trabajo se presenta una herramienta que resuelve todos estos problemas permitiendo la generación automática y masiva de preguntas para la evaluación de algoritmos de obtención de autómatas finitos a partir de expresiones regulares. La evaluación de estos algoritmos es relevante tanto en las asignaturas de lenguajes formales como en las de diseño de compiladores. Con un algoritmo genético se buscan ejercicios de complejidad configurable, tras lo cual se genera el texto, tablas e imágenes asociadas para la carga masiva de todas las preguntas generadas. El principal impacto del uso de esta herramienta está en la enorme reducción del tiempo en la preparación de preguntas, sustituyendo cientos o miles de interacciones con los formularios de Moodle, por unos pocos pasos.

Abstract

Moodle questionnaires are a convenient way to face the online assessment, so relevant in these pandemic times. Unfortunately, their preparation is slow, cumbersome, and prone to mistakes. In addition to designing the question, then it has to be introduced in Moodle. If the question has images, it is necessary to create and upload them to Moodle. It is also necessary to configure the grades associated with the correct answer and the incorrect alternative options. If this process has to be repeated with many questions, the time required ends up being prohibitive. In this work, a tool is presented that solves all these problems, allowing the automatic and massive generation of questions for the evaluation of algorithms for obtaining finite automata from regular expressions. The evaluation of these algorithms is relevant in both formal languages and compiler design subjects. With a genetic algorithm, exercises of configurable complexity are searched, after which the text, tables and associated images are generated for the massive loading of all the questions generated. The main impact of the use of this tool is in the enormous reduction of the time in the preparation of questions, replacing hundreds or thousands of interactions with Moodle forms, for a few steps.

Palabras clave

Moodle, preguntas cloze, cuestionarios automáticos, autómatas finitos, expresiones regulares, algoritmos genéticos.

1. Introducción

Muchas universidades españolas han adoptado Moodle como plataforma de gestión de aprendizaje. Moodle, además de poder utilizarse como repositorio de materiales de asignaturas (con la ventaja adicional de que es posible controlar el acceso que se hace a estos ma-

teriales), ofrece otras funcionalidades. Por ejemplo, la realización de cuestionarios.

Los cuestionarios Moodle son una forma adecuada de evaluar los conocimientos y competencias adquiridos por los alumnos [11]. Se pueden utilizar tanto en la evaluación continua como para la evaluación final de las asignaturas. Moodle ofrece un amplio abanico de tipos de preguntas. Desde las más sencillas, como las de respuesta corta, de cierto o falso, de respuesta múltiple, numéricas o de emparejamiento, a las más complejas y sofisticadas, como las calculadas, las de pinchar y arrastrar texto o imágenes, o las preguntas de tipo cloze. Estas son las más flexibles, permitiendo combinar en una misma pregunta varias opciones de respuesta, como rellenar blancos, preguntas numéricas, selección múltiple de una única respuesta, o selección múltiple de varias respuestas.

La elaboración de los cuestionarios de Moodle puede hacerse de varias formas. La primera, consiste en utilizar los propios formularios proporcionados por Moodle. Pero el proceso es lento y tedioso, y tiene que hacerse pregunta a pregunta. La segunda, permite la carga simultánea de varias preguntas, pero obliga a aprender una sintaxis específica, lo que para muchos docentes puede suponer también un problema. Una tercera alternativa es el uso de herramientas externas a Moodle.

En este artículo se presenta una de estas herramientas, la cual está orientada a la generación de preguntas cloze para alumnos de asignaturas sobre diseño de compiladores o sobre lenguajes formales.

En concreto, facilita enormemente la generación de preguntas sobre los algoritmos de obtención de autómatas finitos a partir de expresiones regulares. Lo único que hay que hacer es introducir la expresión regular y elegir el formato de pregunta de entre un conjunto de plantillas predefinido. No sólo genera el texto de la pregunta, sino también las distintas alternativas de respuesta, y para algunos formatos de pregunta la imagen de los autómatas o árboles de análisis a los que se hace referencia en el texto de la pregunta. Es más, mediante la utilización de algoritmos genéticos, puede generar de forma automatizada el número requerido de preguntas. Lo único que hay que introducir son algunos parámetros sobre la complejidad que se desea tengan las preguntas generadas. Los cuestionarios *cloze* generados por PLQuiz son del tipo particular de preguntas en las que hay que elegir la respuesta correcta entre un conjunto propuesto de alternativas.

El resto del artículo se estructura como sigue: en la Sección 2, para enmarcar el trabajo, se realiza una breve descripción de los conceptos de expresión regular, autómata finito y los algoritmos que permiten obtener los segundos a partir de los primeros. En la Sección 3, se enumeran y describen brevemente algunas otras herramientas que también trabajan con estos conceptos. En la Sección 4, se describe la herramienta PLQuiz, algunos detalles de implementación y se explica su uso mostrando los principales tipos de preguntas generadas. Se finaliza con las conclusiones en la Sección 5.

2. Antecedentes teóricos

Las asignaturas de lenguajes formales o de compiladores generalmente se imparten en el segundo o tercer curso de los grados en informática. Sus contenidos varían entre universidades, pero generalmente comprende lo siguiente:

- Lenguajes de programación: definición formal y base del diseño del lenguaje.
- Especificación formal de lenguajes: los conceptos esenciales de alfabetos, símbolos, lenguajes formales, etc., así como el uso de expresiones regulares y gramáticas.
- Análisis léxico: generalmente con expresiones regulares y herramientas específicas para la construcción de *lexer* (también conocidos como escáneres o *tokenizadores*).
- Análisis sintáctico: análisis de arriba hacia abajo y de abajo hacia arriba, generalmente basado en herramientas específicas para la construcción de analizadores como JavaCC[1], Bison[2], PLY[3], ANTLR[4], etc.

Los primeros contenidos son comunes tanto en las asignaturas sobre autómatas y lenguajes formales como en las de construcción de compiladores y es para estos contenidos para los que la herramienta presentada aquí puede generar cuestionarios de forma automática. En concreto, para evaluar los conocimientos sobre los algoritmos que permiten obtener autómatas finitos a partir de expresiones regulares. En la sección que sigue se da una breve introducción a estos conceptos y algoritmos.

2.1. Expresiones regulares y autómatas finitos

Las expresiones regulares (ER) son un mecanismo para describir tipos especiales de lenguajes formales, conocidos como lenguajes regulares. Utilizando elementos del alfabeto del lenguaje y los operadores de concatenación («$\cdot$», que se suele omitir), de selección («$|$») y de repetición («$\star$») pueden describirse las cadenas pertenecientes al lenguaje. Por ejemplo, la expresión regular `a·(a|b)*·b` representaría el conjun-

[1] https://javacc.org
[2] https://www.gnu.org/software/bison/
[3] http://www.dabeaz.com/ply/
[4] http://www.antlr.org/

to de cadenas de aes y bes que empiezan por *a* y finalizan por *b*.

En el contexto de la construcción de compiladores, las expresiones regulares se utilizan para definir los componentes léxicos que se reconocen en la fase de análisis léxico: identificadores, números en coma flotante, cadenas, etc.

Dada una ER puede construirse un autómata finito (AF) capaz de reconocer las cadenas descritas por la ER. El AF comienza el proceso de reconocimiento en el *estado inicial*, una *función de transición* define cómo, a partir de unos estados puede llegarse a otros. Si la función de transición solo permite que el AF esté en un único estado activo, se trataría de un autómata finito determinista (AFD). Si permite estar en varios[5], sería un autómata finito no determinista (AFND).

Existen algoritmos que permiten la construcción automática de autómatas a partir de una expresión regular. Los implementados en PLQuiz se describen en la siguiente sección.

2.2. Construcción de autómatas a partir de expresiones

Hay varias formas de obtener un AF a partir de una ER. En PLQuiz se generan diferentes tipos de cuestionarios para evaluar el conocimiento sobre los siguientes algoritmos:

- El algoritmo propuesto por McNaughton, Yamada y Thompson [13], que construye un AFND a partir de una expresión regular, analizando de forma recursiva sus partes constituyentes (subexpresiones). El algoritmo crea un AFND para cada sub-expresión, y los une usando transiciones épsilon teniendo en cuenta el operador que las combina.
- El algoritmo de construcción de subconjuntos [16], que permite transformar un AFND en un AFD. Dado que el AFND tiene un número finito de estados, también serán finitas las posibles combinaciones de estados activos (de hecho, algunas de estas combinaciones no se darán en la práctica, al impedir la función de transición que se alcancen), la idea es asociar un estado del AFD con cada una de estas combinaciones y construir una función de transición que refleje como cambian estas combinaciones en el AFND de partida.
- Por último, PLQuiz implementa el algoritmo descrito por Aho, Sethi y Ullman en [1], que utiliza la representación como árbol sintáctico de la expresión regular para calcular tres funciones: *anulable*, *primera-pos* y *última-pos*, que permiten construir una función adicional, *siguiente-pos*, a partir de la cuál puede construirse finalmente la función de transición del AFD que reconoce el lenguaje representado por la ER de partida.

Para profundizar en los conceptos de ER, AF y algoritmos de obtención de AF a partir de ER pueden consultarse los trabajos de Linz [10] y Meduna [14].

3. Trabajos relacionados

La complejidad con la que los alumnos perciben los conocimientos sobre procesadores de lenguajes y sobre autómatas finitos y lenguajes formales ha motivado que muchos docentes hayan desarrollado aplicaciones y simuladores para dar un enfoque más práctico al aprendizaje de dichos conocimientos. Nos restringimos aquí a aquellas que se centran en los aspectos relacionados con el análisis léxico, las ER y los AF. En este grupo, la herramienta que podría considerarse la decana es JFLAP [17]. Está escrita en Java y es muy popular y completa. Permite la construcción y visualización no solo de autómatas finitos, sino de otros dispositivos reconocedores, como los autómatas de pila y las máquinas de Turing. También permite la obtención de un AF a partir de una ER, aunque para esto solo proporciona un algoritmo, que no se corresponde con ninguno de los implementados en PLQuiz. THOTH [7, 8] es otra herramienta, que como JFLAP, permite la creación gráfica de AF e implementa un conjunto más amplio de algoritmos de obtención de AF a partir de ER. Proporciona los algoritmos de Aho-Sethi-Ullman, de McNaughton-Yamada-Thompson y de construcción de subconjuntos, introducidos en la sección previa, además de un tercer algoritmo basado en la utilización de derivadas de las expresiones regulares[6].

Las dos herramientas previas son herramientas de escritorio, pero también existen herramientas que pueden utilizarse sin más que tener un navegador, al ser aplicaciones web. Es el caso de *SELFA-Pro* [5], una herramienta que basa su funcionamiento en el uso de un lenguaje específico del dominio para describir expresiones regulares, autómatas, gramáticas y operaciones sobre los mismos. También es el caso de *Seshat* [2] que permite la visualización paso a paso de algunos de los algoritmos de obtención de AF a partir de ER. *Seshat* utiliza un diseño web adaptativo que permite su utilización también a través de dispositivos móviles. De hecho, las aplicaciones más recientes están dirigidas

[5] Porque de un estado el AF puede evolucionar a varios otros con el mismo símbolo, o porque tiene *transiciones épsilon*, que le permiten cambiar de estado sin necesidad de consumir símbolos de entrada.

[6] Dada una expresión regular, se dice que su derivada respecto a un carácter dado es la expresión regular que describe el lenguaje resultante de eliminar el primer carácter en el subconjunto de todas las cadenas que empiezan por ese carácter en el conjunto definido por la expresión regular de partida [3].

directamente a su uso en móviles. Es el caso de *CM-Simulator* [4] y de *Automata Simulator* [19], ambas orientadas a la construcción, simulación y manipulación de autómatas finitos y no tanto a su obtención a partir de expresiones regulares.

Las anteriores son sólo algunas de las aplicaciones representativas de las disponibles para facilitar el aprendizaje de conocimientos sobre lenguajes formales, ninguna de ellas tiene funcionalidad de generación de cuestionarios. En [20] puede encontrarse una revisión más amplia que incluye también herramientas que cubren otros aspectos de la construcción de compiladores, como los algoritmos de análisis sintáctico. Pero tampoco en esa revisión aparecen herramientas que como PLQuiz estén diseñadas para la generación de automática de cuestionarios.

También existen herramientas y bibliotecas que buscan facilitar la creación de cuestionarios en Moodle, aunque no siempre están bien mantenidas. De momento, no parecen haberse popularizado demasiado y su uso está restringido a los pocos usuarios que conocen de su existencia. Algunos ejemplos serían:

- *Libre-gift*[7], una plantilla de LibreOffice para la creación de cuestionarios en este procesador de texto que luego son exportados a formatos importables desde Moodle.
- *MoodleQUIZ_template_UVa*[8], otra plantilla, en este caso para Word, que facilita la edición de preguntas que luego son exportadas a GIFT, formato que luego puede usarse para importar las preguntas a Moodle.
- *Moodle Cloze and GIFT Code Generator* [21], una macro que permite editar preguntas usando Excel para exportarlas luego al formato GIFT[9].
- *GIFT Quiz Editor*[10], un complemento para los formularios Google que permite su exportación a GIFT.
- *Moodle-questions*[11], una biblioteca para la manipulación de cuestionarios Moodle en Python.
- *Pygiftgenerator*[12], otra biblioteca de Python, esta avalada por una publicación en revista [18], para la generación automática de problemas de física.
- *GIFTGenerator*[13], una herramienta comercial para la generación de cuestionarios.
- *Typeform*[14], otra herramienta comercial para facilitar la creación de cuestionarios.
- R/Exams[15], un paquete para el sistema R enfocado a la generación automática de exámenes que pueden importarse a Moodle. Su desarrollo es bastante activo y cuenta con una importante base de usuarios.

En todo caso ninguna de estas bibliotecas está centrada en la generación de cuestionarios para la evaluación de algoritmos de obtención de autómatas finitos a partir de expresiones regulares[16]. De hecho, a nuestro leal saber y entender, PLQuiz es la única herramienta existente con esta finalidad.

4. La nueva herramienta

El código de la aplicación se ha diseñado para que sea modular y reusable, agrupándolo en paquetes en los que las dependencias se han reducido usando el patrón de diseño *fachada* (*facade pattern*) [6]. Cada paquete principal tiene una estructura de dos niveles: el primer nivel contiene las clases *fachada*, visibles al resto de la aplicación, mientras que el segundo contiene la lógica y estructuras de datos internas, utilizadas solo por el paquete padre. Aunque el lenguaje principal en la que está escrita es Java, también se utilizan otros, como XML para la generación de los cuestionarios Moodle, o LaTeX y TikZ para la obtención de ejercicios para exámenes escritos.

La aplicación se encuentra disponible en Github (https://github.com/RobertoIA/PLQuiz).

4.1. Generación de expresiones regulares

Aunque las ER se especifican inicialmente como cadenas de caracteres, se representan internamente como árboles binarios. Como la generación de árboles es un tema muy estudiado en programación genética, esto nos permite utilizar directamente algoritmos existentes para la generación de nuevas expresiones.

El algoritmo concreto implementado es el del método «*full*» [9]. Se generan árboles binarios en los que cada nodo no hoja tiene 1 o 2 hijos, y en el que todas las hojas se encuentran a la misma profundidad. El algoritmo restringe las etiquetas que pueden usarse en cada nodo generado, en función de su profundidad. Los nodos símbolo solo se permiten a la profundidad máxima, mientras que un nodo cerradura se permite a cualquier profundidad, excepto la máxima. Se utilizan restricciones adicionales a la hora de generar los hijos de un nodo, en función del operador que se usa para el nodo (un nodo cerradura solo podrá tener un hijo,

[7] https://github.com/clopezno/libre-gift
[8] https://github.com/juacas/MoodleQUIZ_template_UVa
[9] https://hbubecc.wixsite.com/jordan/tools
[10] https://bit.ly/2Zaohsc
[11] https://github.com/gethvi/moodle-questions
[12] https://gitlab.com/EHU/pygiftgenerator
[13] https://www.giftgeneratorapp.com/
[14] https://www.typeform.com/quizzes/

[15] http://www.r-exams.org
[16] R/exam tiene una plantilla para generar ejercicios para evaluar la interpretación de un diagrama de autómata, http://www.r-exams.org/templates/automaton/, pero no hemos podido encontrar ninguna para evaluar algoritmos de obtención de AF a partir de ER.

mientras que los nodos concatenación o selección, podrán tener dos).

Con estas restricciones, ya es posible generar una población inicial de ER aleatorias. En el proceso selectivo de los mejores «individuos» se tienen en cuenta consideraciones adicionales: *i*) número de símbolos presentes en la ER (sin contar el que representa la palabra vacía, *ii*) número de estados en la función de transición del AF obtenido a partir de la ER, *iii*) presencia o ausencia del símbolo ε, el cual representa la palabra vacía.

Una vez disponemos de una población aleatoria inicial, el objetivo del algoritmo genético es la obtención de soluciones similares en su camino hacia el óptimo. En el campo de la programación genética, de donde obtenemos nuestros métodos, se considera que los operadores de mutación y reproducción tienden a producir resultados similares [12]. En este caso, utilizamos un operador de mutación, consistente en reemplazar un sub-árbol de la ER por uno generado aleatoriamente, por resultar más sencillo en cuanto a implementación.

Dado que el objetivo del generador es producir ER que resulten en problemas que no sean ni excesivamente complejos ni triviales, el espacio de búsqueda de los algoritmos puede limitarse de tal forma que produzca el mayor número de expresiones posible dentro de este rango. Una vez definida la complejidad deseada en función del número mínimo y máximo de símbolos y estados distintos en las expresiones resultado, se utiliza el método de generación de árboles descrito anteriormente para construir miles de árboles de distintas profundidades. De este modo se determina experimentalmente qué rango de profundidades de árbol produce mayor proporción de árboles de la complejidad deseada para cada tipo de problema. En este caso, se comprobó que para los problemas basados en Aho-Sethi-Ullman resultaba apropiada una profundidad de árbol entre 3 y 6, y para los problemas de McNaughton-Yamada-Thompson, era mejor una profundidad entre 1 y 5.

Trabajar con un conjunto de posibles soluciones bien definido nos permite comparar distintos métodos de generación de problemas de manera empírica, así como elegir en cada caso el método más adecuado al problema. Las pruebas realizadas permitieron comparar el algoritmo genético implementado, con una generación puramente aleatoria, usando el método de inicialización hasta obtener un resultado aceptable. Se comprobó que los problemas de McNaughton-Yamada-Thompson se generaban de manera más eficiente con el algoritmo genético completo, mientras que en los de tipo Aho-Sethi-Ullman resultaba más rápido utilizar una generación aleatoria. Una posible causa es que, para las restricciones fijadas, el espacio de búsqueda para problemas de Aho-Sethi-Ullman contiene más potenciales soluciones, y por lo tanto, requiere menor carga de optimización.

4.2. Generación de documentos LaTeX

Aunque el objetivo principal de PLQuiz es la generación de cuestionarios Moodle, adicinalmente permite la obtención de documentos LaTeX de ejercicios, lo que facilita también la realización de exámenes escritos en los que poder incluir la evaluación de los algoritmos de obtención de AF. Los documentos LaTeX [15] usan la clase de documento exam[17] e incluyen tanto los enunciados de los ejercicios como su solución. Lo único que hay que hacer para mostrar o no la solución es cambiar una de las opciones de la clase de documento. De esta forma, se puede obtener tanto el documento para el examen como el documento de solución para su publicación posterior. Los árboles y los diagramas de transición de los AF de los enunciados o de las soluciones se obtienen de forma automática generando código TikZ [22]. Al igual que con los cuestionarios de Moodle, el ahorro de trabajo para el profesor es importante, ya que la obtención manual de estos diagramas requiere de bastante tiempo. En la Figura 1 pueden verse dos ejemplos de estos ejercicios, en los que se muestra tanto el enunciado como la solución.

4.3. El uso de la herramienta

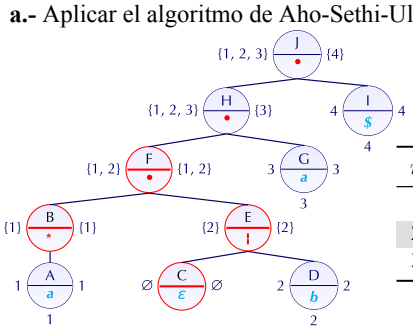
Las principales utilidades de la herramienta son: la generación de cuestionarios importables a Moodle, y la generación de funcionalidades que facilitan la creación y gestión de dichos cuestionarios. También permite obtener documentos LaTeX con los ejercicios generados.

La herramienta desarrollada permite añadir y eliminar preguntas. Durante el proceso de adición, puede elegirse el tipo de ejercicio para el que se generará dicha pregunta: bien el basado en obtener un AFND con McNaughton-Yamada-Thompson para a continuación utilizar construcción de subconjuntos, o la obtención directa de un AFD utilizando el algoritmo de Aho-Sethi-Ullman. Por otro lado, existe la posibilidad de añadir en una única operación bloques de múltiples problemas de cualquiera de los tipos (Figura 2), reduciendo la similitud de las preguntas a través de un rango de variación, y generando automáticamente tanto las ER como los diagramas y tablas resultado de aplicar los algoritmos de obtención de AF. En cuanto a la operación de eliminación, pueden elegirse los ejercicios concretos que quieren eliminarse o eliminar todos para comenzar la creación del cuestionario desde cero.

Cada tipo de pregunta (Aho-Sethi-Ullman o McNaughton-Yamada-Thompson) dispone de tres subtipos de ejercicios distintos, que se corresponden

[17] https://ctan.org/pkg/exam

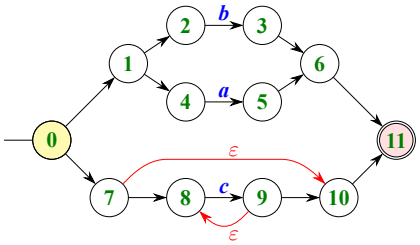
a.- Aplicar el algoritmo de Aho-Sethi-Ullman a la expresión regular: $a^* \cdot (\varepsilon|b) \cdot a$



n	stePos(n)
1	**1, 2, 3**
2	**3**
3	**4**

$\mathcal{Q}$	a	b	*Posiciones*
A	**B**	**C**	**1, 2, 3**
(B)	**B**	**C**	**1, 2, 3, 4**
C	**D**	**E**	**3**
(D)	**E**	**E**	**4**
E	**E**	**E**	∅

b.- Completa la tabla de función de transición para el AFD que se obtendría de aplicar el método de construcción de subconjuntos al AFND de la figura



$\mathcal{Q}$	a	b	c	*Estados del AFND*
(A)	**B**	**C**	**D**	**0-2, 4, 7, 8, 10, 11**
(B)	**E**	**E**	**E**	**5, 6, 11**
(C)	**E**	**E**	**E**	**3, 6, 11**
(D)	**E**	**E**	**D**	**8-11**
E	**E**	**E**	**E**	∅

Figura 1: Ejemplos de ejercicios en LaTeX para inclusión en exámenes escritos: *a*) Aplicación de Aho-Sethi-Ullman para obtener un AFD a partir de una expresión regular, *b*) Aplicación de «Construcción de subconjuntos» para obtener un AFD a partir del diagrama de transición de un AFND.

con la aplicación de distintos algoritmos o pasos parciales de algoritmo. Estos son para los ejercicios de Aho-Sethi-Ullman: «*construcción de árbol*», «*etiquetado de árbol*» y «*construcción de tablas*», y para los ejercicios de McNaughton-Yamada-Thompson: «*construir autómata*», «*resolver expresión*» y «*resolver autómata*». Cuando se selecciona un subtipo, cambiará automáticamente el tipo de ejercicio mostrado, o bien el que se mostrará al generar el ejercicio, si no se ha introducido ninguna expresión.

A la hora de generar una pregunta, podemos definir, entre otras cosas: *i*) si la expresión regular asociada incorporará transiciones vacías (ε), *ii*) el número de símbolos contenidos en la *expresión regular*, *iii*) el número de estados en la tabla de transición del problema resuelto. A partir de este punto, se pueden generar expresiones, comenzando en ese momento la búsqueda de una expresión regular que produzca dicha pregunta.

Una vez configuradas las preguntas, pueden exportarse al formato *Moodle XML*, para poder ser importadas a continuación al banco de preguntas de Moodle. En el archivo generado se incluyen también todas las imágenes necesarias para la realización del ejercicio.

En las figuras 3 a 4 pueden verse algunos tipos de ejercicios, tal como aparecerían en Moodle. En estos ejercicios, los alumnos tienen que completar las tablas resultado de la aplicación del algoritmo, eligiendo para cada entrada de la tabla la opción correcta, de entre un conjunto de opciones disponibles en un desplega-

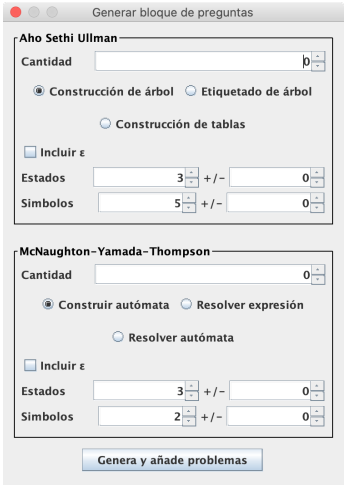


Figura 2: Diálogo para la generación en bloque de varias preguntas.

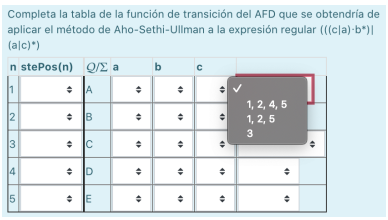


Figura 3: Tipo de ejercicio «Construcción de tablas» para el algoritmo de Aho-Sethi-Ullman.

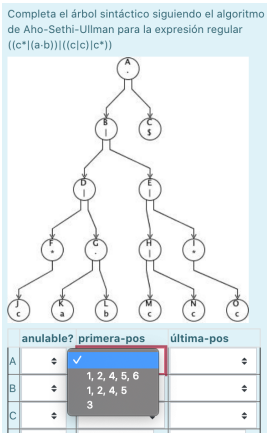


Figura 4: Tipo de ejercicio «Etiquetado de árbol» para la evaluación del algoritmo de Aho-Sethi-Ullman.

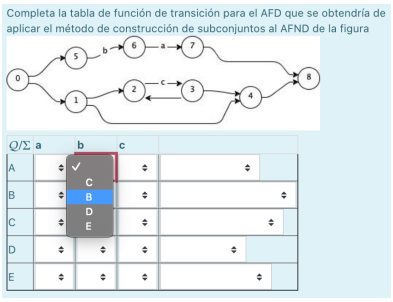


Figura 5: Tipo de ejercicio «Resolver autómata» donde el alumno tiene que aplicar el algoritmo de construcción de subconjuntos.

ble. La realización a mano de este ejercicio implicaría tener que configurar, para cada una de las entradas de la tabla, todas las alternativas, lo que sería bastante tedioso y donde inevitablemente cometeríamos errores.

4.4. Valoración de la herramienta

Para aportar cierta perspectiva sobre la conveniencia de la herramienta, en esa sección se aporta la experiencia personal sobre el uso de la misma.

Desde hace tres cursos, los cuestionarios de evaluación continua de la asignatura «Procesadores de Lenguajes» incorporan ejercicios generados con PLQuiz. Desde nuestro punto de vista, la principal ventaja de PLQuiz es la facilidad de poder generar numerosos ejercicios de forma automática. Estos ejercicios pueden agruparse en categorías del banco de preguntas de Moodle que pueden utilizarse para añadir preguntas aleatorias a los cuestionarios. Dado que PLQuiz genera seis tipos de ejercicios distintos, puede tenerse una categoría para cada uno de ellos, permitiendo añadir hasta seis preguntas aleatorias diferentes a los cuestionarios. Este proceso de aleatorización minimiza el riesgo de que algunos alumnos se sientan tentados a compartir las respuestas a los ejercicios, dado que con un número suficiente de ejercicios, las posibilidades de que a dos alumnos les toquen las mismas preguntas son realmente bajas. Además, dentro de cada una de estas categorías, tenemos subcategorías adicionales que vamos rotando cada curso, de modo que las preguntas de un un curso no coincidan con las de los dos cursos previos.

Para tener una idea objetiva del ahorro de tiempo que supone la generación automática de preguntas con PLQuiz, analicemos lo que ocurre con el tipo de ejercicio «Construcción de árbol» para Aho-Sethi-Ullman. En este ejercicio al alumno se le muestra la expresión regular de partida, así como varios árboles entre los que tiene que elegir el que se corresponde con la expresión regular mostrada. Si quisiéramos introducir de forma manual esta pregunta utilizando los formularios de edición de preguntas en Moodle, este trabajo supondría realizar ¡hasta 50 interacciones! A esto habría que sumar el tiempo de preparar el dibujo de los cuatro árboles para poder subirlos en la parte de respuestas correcta y alternativas. Incluso siendo ágil en la creación de los árboles, esto puede suponer fácilmente al menos media hora para tener una pregunta. En el mismo tiempo, usando PLQuiz podemos generar y subir a Moodle decenas de preguntas listas para ser importadas aleatoriamente en un cuestionario.

Cuando la preparación de las preguntas lleva mucho tiempo, uno tiende a adoptar una actitud protectora con los ejercicios que se tienen preparados, intentando reservarlos para las pruebas de evaluación y con cierta reticencia a que puedan circular y ser compartidos entre los alumnos. Con PLQuiz no es necesario este secretismo, dado que es realmente sencillo generar nuevos ejercicios. Por ejemplo, el curso pasado, además del banco de preguntas para la evaluación continua, se generaron preguntas adicionales para cuestionarios de autoevaluación que los alumnos podían aprovechar para prepararse de cara al examen. Estos cuestionarios tuvieron muy buena acogida, a pesar de ser totalmente opcionales. De hecho, el 75 % de los alumnos decidieron realizarlos. Lamentablemente, no contamos con evidencias objetivas adicionales de la utilidad de estos cuestionarios, aunque en futuros cursos esperamos cruzar las calificaciones finales de los alumnos con la realización de estos cuestionarios, esperando encontrar alguna correlación positiva.

5. Conclusiones

Hemos presentado una herramienta capaz de generar cuestionarios de Moodle, de forma masiva (en gran número) y automática, para la evaluación de algoritmos de obtención de autómatas finitos a partir de expresio-

nes regulares. Los ejercicios pueden variar en complejidad, gracias a la aplicación de un algoritmo genético. Una vez definidos los ejercicios, puede generarse el texto, tablas e imágenes asociados. Estos ejercicios se pueden exportar para incorporar al banco de preguntas de Moodle, o a documentos LaTeX para su utilización en pruebas escritas. Además, las preguntas pueden utilizarse en otros LMS, dado que muchos ofrecen la posibilidad de importar preguntas en el formato de Moodle.

Esta herramienta puede tener un impacto importante tanto en las asignaturas de lenguajes formales como en las de diseño de compiladores, sobre todo por la reducción del tiempo dedicado a la preparación de preguntas.

Referencias

[1] Alfred V Aho, Ravi Sethi, y Jeffrey D Ullman. Compilers, principles, techniques. *Addison Wesley*, 7(8):9, 1986.

[2] Álvar Arnaiz-González, Jose-Francisco Díez-Pastor, Ismael Ramos-Pérez, y César García-Osorio. Seshat—a web-based educational resource for teaching the most common algorithms of lexical analysis. *Comput. Appl. Eng. Educ.*, 26(6):2255–2265, 2018.

[3] Janusz A. Brzozowski. Derivatives of regular expressions. *J. ACM*, 11(4):481–494, octubre 1964.

[4] Daniela Chuda, Jakub Trizna, y Peter Kratky. Android automata simulator. En *Proceedings of the International Conference on e-Learning*, páginas 80–4, 2015.

[5] Jesús Gallardo Casero, José Jesús Castro Sánchez, y Raúl Miguel Sabariego. Experiencias de uso y evaluación de una herramienta de apoyo a la enseñanza de teoría de autómatas y lenguajes formales. En *Actas de las XXII JENUI*, páginas 327 – 334. Universidad de Almería, 2016.

[6] Erich Gamma, Richard Helm, Ralph Johnson, y John M. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1ª edición, 1994.

[7] César García-Osorio, Andrés Arnaiz-Moreno, y Álvar Arnaiz-González. THOTH: A new tool for automata theory learning. En *International Technology, Education and Development Conference*. IATED, Marzo 2007.

[8] César García-Osorio, Iñigo Mediavilla-Sáiz, Javier Jimeno-Visitación, y Nicolás García-Pedrajas. Teaching push-down automata and turing machines. En *Proceedings of the 13th annual conference on Innovation and technology in computer science education*, páginas 316 – 316, 2008.

[9] John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.

[10] Peter Linz. *An introduction to formal languages and automata*. Jones & Bartlett Publishers, 6ª edición, 2017.

[11] José M López, Eva Romero, y Eva Ropero. Utilización de Moodle para el desarrollo y evaluación de competencias en los alumnos. *Formación universitaria*, 3(3):45–52, 2010.

[12] Sean Luke y Lee Spector. A revised comparison of crossover and mutation in genetic programming. En *Proceedings of the Third Annual Conference on Genetic Programming 1998*, páginas 208–213, University of Wisconsin, Madison, Wiscosin, USA, 1998. Morgan Kaufmann.

[13] Robert McNaughton y Hisao Yamada. Regular expressions and state graphs for automata. *IRE Trans Electron Comput*, 9(1):39–57, 1960.

[14] Alexander Meduna. *Formal languages and computation: models and their applications*. CRC Press, 2014.

[15] Frank Mittelbach, Michel Goossens, Johannes Braams, David Carlisle, y Chris Rowley. *The LATEX companion*. Addison-Wesley Professional, 2004.

[16] Michael O Rabin y Dana Scott. Finite automata and their decision problems. *IBM J Res Dev*, 3(2):114–125, 1959.

[17] Susan H Rodger y Thomas W Finley. *JFLAP: an interactive formal languages and automata package*. Jones & Bartlett Learning, 2006.

[18] Jon Sáenz, Idoia G Gurtubay, Zunbeltz Izaola, y Gabriel A López. Pygiftgenerator: a python module designed to prepare moodle-based quizzes. *Eur J Phys*, 42(1):015702, 2020.

[19] Tuhina Singh, Simra Afreen, Pinaki Chakraborty, Rashmi Raj, Savita Yadav, y Dipika Jain. Automata simulator: A mobile app to teach theory of computation. *Comput. Appl. Eng. Educ.*, 27(5):1064–1072, 2019.

[20] Srećko Stamenković, Nenad Jovanović, y Pinaki Chakraborty. Evaluation of simulation systems suitable for teaching compiler construction courses. *Comput. Appl. Eng. Educ.*, 28(3):606–625, 2020.

[21] Jordan Svien. Improvements to the moodle cloze and GIFT code generator. En *Proceedings of MoodleMoot Japan 2019 Annual Conference*, páginas 19–23, 2019.

[22] Tilll Tantau. The TikZ and PGF packages (manual for version 3.0. 1a; 2015.08. 29). Technical report, Institut für Theoretische Informatik Universität zu Lübeck, 2015.