Reorganización de las prácticas de compiladores para mejorar el aprendizaje de los estudiantes

Jaime Urquiza-Fuentes, Francisco Almeida-Martínez, Antonio Pérez-Carrasco
Departamento de Lenguajes y Sistemas Informáticos I, LITE – Lab. de Tecnologías de la Información en la Educación
Universidad Rey Juan Carlos
C/Tulipán s/n, 28933 (Móstoles) Madrid
{jaime.urquiza,francisco.almeida,antonio.perez.carrasco}@urjc.es

Resumen

La parte práctica de asignaturas como Compiladores o Procesadores de Lenguajes (las trataremos como la misma en el resto de la comunicación) suele ser bastante costosa, ya que requiere cierto grado de planificación y continuidad en el trabajo de los estudiantes y las herramientas utilizadas no tienen una conexión clara con los fundamentos teóricos. Nuestra propuesta estructura estas sesiones prácticas en tres tipos: las que se encargan de enlazar teoría y práctica, las que introducen a los estudiantes las herramientas de generación de compiladores y la final donde se desarrolla un compilador de cierta complejidad. Con este enfoque hemos mejorado el porcentaje de éxito en la parte práctica hasta un 86%.

1. Introducción

Compiladores es una de las asignaturas más difíciles de las titulaciones de Informática. Aparte de basarse en la teoría de lenguajes formales, cuyo grado de abstracción añade algo de complejidad, la parte práctica suele ser bastante compleja y costosa.

Esta práctica suele durar la mayor parte del curso requiriendo cierto grado de planificación y continuidad en el trabajo de los estudiantes. Por otro lado ocurre que las herramientas utilizadas en la práctica no tienen una conexión clara con los fundamentos teóricos. En esta comunicación presentamos una reorganización de las prácticas que ayude a los estudiantes a superar estos problemas.

En la siguiente sección describimos los distintos enfoques que se han utilizado en la enseñanza de la asignatura. A continuación detallamos el contexto educativo de la propuesta. En la sección 4 especificamos las nuevas prácticas integradas en el curso, su tipología y contenidos. Finalmente, en las secciones 5 y 6 exponemos los resultados de esta propuesta así como las conclusiones.

2. Trabajos relacionados

La enseñanza de estas asignaturas se realiza de formas muy variadas. Desde el punto de vista de los contenidos, hay enfoques centrados en la teoría de construcción con una cobertura variada de cada concepto –por ejemplo en cuanto a algoritmos de análisis sintáctico o detalle en la implementación de la tabla de símbolos. Otros enfoques se centran en el diseño de lenguajes de programación [1], o incluso en la ingeniería del software. Waite W.M. [16] describe varios de estos enfoques.

Otro punto importante es la práctica necesaria en estas asignaturas. El enfoque típico es una práctica que dura todo el curso, pero se han planteado distintas alternativas evitando tener que crear un compilador entero, por ejemplo, desarrollar diferentes partes de uno [4] cuya estructura ya está desarrollada, hacer pequeñas prácticas que no tienen porqué estar relacionadas entre sí [14] o incluso explorar el comportamiento de un compilador real con una herramienta de depuración [17].

Aún así, la construcción de un compilador completo sigue siendo interesante [7]. Para esta opción también existen distintas variantes, aunque todas siguen un desarrollo incremental similar: análisis léxico, sintáctico, semántico y generación, incluyendo a veces el desarrollo de una máquina virtual. Sathi, H.L. [14] disminuye la complejidad del compilador a cambio de no usar herramientas generadoras. Aho, A.V. [1] pide a los estudiantes que diseñen su propio lenguaje para el que construirán el compilador. Aycock [3] se centra en que los estudiantes puedan probar sus

compiladores con programas complejos y para ellos desarrollan lenguajes especiales. Finalmente, otra variante utiliza lenguajes fuente de ámbitos distintos a la programación [6][8][13][18].

Nosotros hemos decidido implantar las prácticas basadas en el desarrollo de una parte significativa de un compilador a lo largo de todo el curso. El trabajo lo realizan en grupo, así pueden afrontar a la complejidad de un lenguaje más realista. Además, estas prácticas aportan una visión completa del compilador, y de la relación entre las distintas etapas.

3. Contexto educativo

En esta sección describimos la organización de nuestra asignatura y revisamos el rendimiento de los estudiantes en la práctica del curso completo.

Nuestra asignatura es anual y troncal (todos los estudiantes de la titulación deben aprobarla). Sus contenidos se estructuran de la forma típica. Una introducción general y el análisis léxico y sintáctico en la primera mitad del curso, y la traducción dirigida por sintaxis, el análisis semántico y la generación de código intermedio y final en la segunda.

En nuestra universidad existen dos periodos de evaluación oficiales –junio y septiembre–, por lo que la práctica tiene 4 entregas: dos parciales para el analizador léxico y sintáctico y dos finales coincidiendo con los periodos de evaluación oficiales donde se debe completar el resto de la práctica. Antes de realizar ninguna entrega, los estudiantes deben organizarse en grupos de como máximo tres personas.

Dado que en nuestra titulación no se puede asegurar que nuestros estudiantes posean conocimientos suficientes de lenguaje ensamblador, la práctica no exige la generación de código objeto. En su lugar usamos lenguajes finales para los que el esfuerzo de traducción a realizar sea significativo, por ejemplo lenguajes de medio-bajo nivel con restricciones similares a las existentes en los lenguajes ensambladores (expresiones aritméticas con un máximo de dos operandos, sin bucles).

3.1. Retrospectiva de las prácticas

Hemos analizado el rendimiento de los estudiantes en las distintas entregas. En las gráficas correspondientes (Figura 1, Figura 2 y Figura 3), las cantidades reflejan el porcentaje de los grupos registrados que realizaron de forma satisfactoria cada entrega: AL–análisis léxico, AS–sintáctico, TDS(J)–resto de la práctica en junio y TDS(S)–resto de la práctica en junio y septiembre acumulados.

No hemos tenido en cuenta en este análisis el primer año de impartición de la asignatura, curso 2000-01, por la poca cantidad de estudiantes. Tampoco hemos tenido en cuenta los cursos 2004-05 y 2005-06, ya que carecemos de los datos necesarios para el análisis. Así pues esta retrospectiva contempla 5 cursos académicos en dos periodos diferentes, 2001-04 y 2006-08. Aplicamos la propuesta detallada en esta comunicación en el curso 2008-09.

Durante los *3 primeros años*, el enunciado de la práctica mantenía una parte básica para todos y a su vez permitía numerosas variantes usando opciones en cuanto a técnicas de compilación –analizadores sintácticos utilizados y representaciones de código intermedio– y características del lenguaje fuente a compilar –tipos de datos complejos, sentencias de control de flujo. La asignación de las opciones a cada grupo era aleatoria. Para aprobar la práctica el grupo debía implementar las características que se les había asignado, los analizadores léxico y sintáctico junto con al menos un tipo de datos y una sentencia de control de flujo. Los incrementos de la calificación se obtenían a través de la calidad de la memoria y la ampliación de las características implementadas.

El lenguaje fuente que utilizamos fue cambiando, el primer año en que se impartió la asignatura, 2000-01, utilizamos *Java*. Después de observar los problemas que tuvieron los estudiantes decidimos cambiarlo. Así, durante los tres primeros años del periodo analizado, 2001-04, los lenguajes fuentes fueron: un lenguaje de consultas a bases de datos estilo SQL que había que traducir a operaciones de álgebra relacional, y dos variantes léxico-sintácticas del lenguaje C con restricciones semánticas como la eliminación de operaciones de punteros.

Como se puede ver en la Figura 1, en los tres años la cantidad de grupos que completó con éxito la entrega del analizador léxico fue bastante importante, una media de 76,9%.

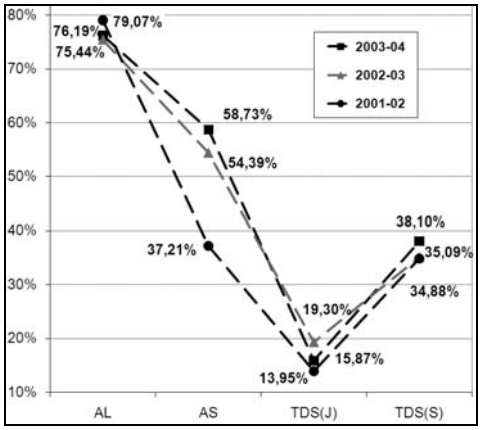

Figura 1. Gráfica de los años 2001-04

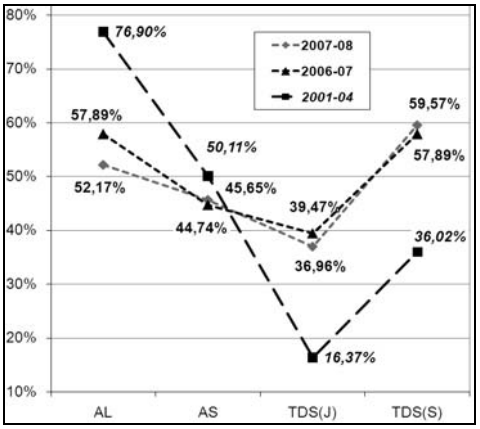

Figura 2. Gráfica de los años 2006-08. Los datos de los tres años anteriores, 2001-04, se han resumido con las medias de cada entrega

La cantidad de grupos que abandonaron la práctica después de esa entrega fue muy alta, especialmente el primer año donde entregaron el analizador sintáctico sólo un 37,21%. En los otros dos años estuvo alrededor del 56%. Aún así la cantidad de grupos que continuaban con la práctica siguió disminuyendo, terminaron con éxito la entrega de junio una media del 16,37%. A final de curso, terminó la práctica una media del 36,02% de los grupos registrados.

La tasa de abandono fue demasiado alta. Entre las principales causas encontramos una típica, el proyecto de compiladores es el primero de larga duración que afrontan los estudiantes de nuestra titulación [14][15]. A pesar de la estructuración en entregas a los estudiantes les costaba establecer objetivos claros. Además, el hecho de trabajar con enunciados distintos les impedía compartir ideas sobre soluciones a problemas que pudieran encontrar.

En el siguiente periodo analizado tratamos de solventar estos problemas. En primer lugar permitimos la selección voluntaria de las opciones de compilación y características del lenguaje, e introdujimos la defensa presencial para evitar el plagio entre grupos. En segundo lugar definimos unos criterios totalmente detallados que permitían establecer objetivos claros para alcanzar las calificaciones de aprobado, notable o sobresaliente.

Esta vez los lenguajes fuente eran orientados a objetos. El primer año se pidió traducir un subconjunto significativo de Java, llamado *SimpleJava*, a un subconjunto de SmallTalk, llamado *TinyTalk*. El segundo año se invirtió el enunciado. La complejidad de los lenguajes fuente seguía siendo similar a otros años requiriendo el uso de análisis semántico, tablas de símbolos, ámbitos, etc.

En la Figura 2 mostramos los datos de este periodo, 2006-08, junto con el periodo anterior, 2001-04, resumido por sus medias. Aunque en un principio pudiera parecer que empeoramos resultados, ya que los porcentajes de la primera entrega eran menores que en años anteriores, los resultados en la segunda entrega se acercaron al periodo anterior y mejoraron en las entregas de junio y septiembre aproximadamente en un 20%. Creemos que mejoramos la capacidad de los grupos para valorar sus posibilidades a la hora de terminar con éxito la práctica, ya que la tasa de abandono entre entregas es mucho menor, y en cómputo global la cantidad de grupos que hicieron la primera entrega es prácticamente igual a la de grupos que terminaron la práctica en septiembre.

Sin embargo, el porcentaje de grupos que termina la práctica a final de curso sigue siendo bajo, una media de 58,73% en este periodo. En términos de porcentajes de alumnos que aprueban la práctica con respecto a los matriculados, en el primer periodo tuvimos una media de 33,2% y en el segundo 52,43%. Comparando estas cifras con

los porcentajes de estudiantes que aprueban la parte teórica, una media del 85,05%, la diferencia salta a la vista.

A la luz de estos datos creemos que debemos establecer mejor el vínculo entre los conceptos teóricos, que los estudiantes sí dominan, y los prácticos. Para ello proponemos una reorganización de las sesiones prácticas de la asignatura que refuerce ese vínculo. Aplicamos la organización en el curso 2008-09.

4. Reorganización de la prácticas

Nuestra propuesta es salvar el salto entre la teoría –algoritmos y estructuras formales– y la práctica –herramientas de generación e implementación de estructuras de datos complejas– con prácticas más sencillas y cercanas entre ambos extremos. Vamos a integrar en la asignatura otros dos tipos de prácticas: básicas y aplicativas, a parte de la práctica final.

Las *prácticas básicas* tienen carácter voluntario y tratan de acercar los conceptos teóricos a la práctica de dos formas diferentes: mostrando el funcionamiento de los algoritmos vistos en clase mediante herramientas de simulación/visualización o relacionando el resultado producido por las herramientas de generación con sus fundamentos teóricos. La duración será de 1 o 2 horas. El resultado final de este tipo de prácticas es que el estudiante ha visto en funcionamiento los algoritmos, los ha relacionado con su implementación por parte de las herramientas generadoras y ha utilizado estas una forma básica y tutorizada.

Las *prácticas aplicativas* son voluntarias e incentivadas. Pueden aumentar 1 punto (sobre 10) la nota final de la asignatura, siempre y cuando se apruebe la parte teórica. Estas prácticas tratan de involucrar a los estudiantes en el uso de las herramientas de generación resolviendo problemas algo más complejos. La duración será de 3 a 15 días. El resultado final de este tipo de prácticas es que los estudiantes aprenden cuándo y cómo aplicar las funcionalidades de estas herramientas, lo que les ayudará a afrontar la realización de la práctica final.

A continuación describimos las prácticas agrupadas en las tres entregas de la práctica final: análisis léxico, sintáctico y traducción dirigida por la sintaxis (análisis semántico y generación de código).

4.1. Análisis léxico

Durante la fase de análisis léxico planificamos tres prácticas, dos básicas y una aplicativa.

La primera práctica básica es de simulación y trata sobre *autómatas y expresiones regulares*. El objetivo de esta práctica es mostrar el funcionamiento de los fundamentos teóricos del análisis léxico. Usamos la herramienta JFlap [12] para que los estudiantes generen autómatas finitos y expresiones regulares y experimenten con el proceso de reconocimiento.

La segunda práctica básica es sobre herramientas generadoras y trata sobre la *herramienta JFLex* [10]. El objetivo de esta práctica es introducir a los estudiantes en la utilización de los generadores de analizadores léxicos. Lo enfocamos como un tutorial sobre la herramienta con ejercicios simples, así los estudiantes terminan conociendo las distintas posibilidades de JFlex. También se les muestra el código generado identificando las tablas de transición del autómata finito. De esta forma mostramos la aplicación directa de los conceptos teóricos en la práctica.

Finalmente, la práctica aplicativa consiste en implementar un *traductor de código Morse*. El objetivo de esta práctica es que los estudiantes apliquen las posibilidades de traducción de JFlex a un enunciado pseudo-realista. Se les proporciona una descripción del código Morse y se les pide que construyan un traductor a/desde caracteres del alfabeto occidental. El plazo de entrega fueron dos días, debiendo proporcionar la especificación JFlex del traductor. El peso en la nota extra de prácticas era de un 10%.

Con estas prácticas conseguimos que los estudiantes se ejerciten en el diseño de autómatas y expresiones regulares, vean la conexión con un generador de analizadores léxicos y practiquen con él antes de afrontar la parte del analizador léxico correspondiente a la práctica final.

4.2. Análisis sintáctico

Durante la fase de análisis sintáctico planificamos ocho prácticas, seis básicas y dos aplicativas. Utilizamos la herramienta JFlap [12] para el trabajo con gramáticas y autómatas, así como con

analizadores LL(1) y SLR(1). Las herramientas generadoras que usamos eran ANTLR [11] para analizadores descendentes y CUP [9] para analizadores ascendentes.

La primera práctica básica era de simulación y trataba de *autómatas con pila (AP) y gramáticas independientes del contexto (GIC)*. El objetivo de esta práctica es mostrar el funcionamiento de los fundamentos teóricos del análisis sintáctico. Los ejercicios son las típicas descripciones de lenguajes para los que hay que diseñar el AP y la GIC que lo reconozca. En el caso de los APs, la comprobación de su funcionamiento es inmediata. Sin embargo, en el caso de las GICs la herramienta debe construir el analizador correspondiente, en esta práctica utilizamos el analizador con retroceso. JFlap muestra la ejecución de este algoritmo informando sobre la cantidad de nodos creados durante la ejecución, de esta forma conseguimos que los estudiantes sean conscientes de la necesidad de encontrar otros algoritmos de construcción de analizadores sintácticos más eficientes.

Las dos siguientes prácticas básicas tratan de analizadores descendentes, la primera es de simulación y la segunda de generador. En primer lugar usamos JFlap para experimentar con los conceptos teóricos relacionados con los *analizadores LL(1)*. Esta herramienta permite a los estudiantes explorar de forma activa el proceso de construcción de los conjuntos *cabecera* y *siguientes* de cada símbolo no terminal, así como la tabla de análisis LL(1). La exploración activa consiste en pedir al estudiante que especifique el contenido de los conjuntos y las celdas de la tabla, indicando los errores cometidos y proporcionando las soluciones correctas si así lo necesita el estudiante. Finalmente JFlap construye el analizador y permite que el estudiante visualice su comportamiento usando sus propias cadenas de entrada.

La *práctica básica de generación* trata sobre ANTLR [11]. El objetivo de esta práctica es doble. Por un lado queremos introducir a los estudiantes en el uso de generadores de analizadores sintácticos descendentes de carácter profesional como ANTLR [11]. Por otro lado los estudiantes podrán ver la aplicación práctica de los conceptos teóricos relacionados. De nuevo utilizamos un enfoque de tutorial sobre la herramienta, construyendo analizadores para lenguajes simples como los usados en la práctica sobre APs y GICs. Finalmente los estudiantes explorarán el código fuente generado por ANTLR identificando conceptos teóricos como las reglas borradoras, los símbolos de anticipación (y su construcción con los conjuntos *cabecera* y *siguientes*), los métodos asociados a no terminales o la detección de terminales en la cadena de entrada.

Para terminar con los analizadores descendentes usamos una *práctica aplicativa sobre ANTLR*. El objetivo de esta práctica es que los estudiantes diseñen gramáticas no triviales teniendo en cuenta las restricciones LL(1). Proponemos a los estudiantes una gramática de expresiones aritméticas sin precedencia alguna en los operadores que deben modificar para que los operadores aritméticos cumplan unas normas de precedencia específicas. Además les pedimos que documenten la solución con visualizaciones que justifiquen la existencia de estas precedencias en los árboles sintácticos. Las herramientas de visualización utilizadas son VAST [2] y ANTLRWorks [4]. En concreto, los estudiantes deben modificar la gramática, producir visualizaciones seleccionando ellos mismos las cadenas de entrada y explicar dichas visualizaciones. Además se les pide generar visualizaciones de ejemplo para las situaciones erróneas LL(1): fallo con el símbolo director y terminal no encontrado en la cadena de entrada. El plazo de entrega fue de tres días, debiendo proporcionar la especificación ANTLR de la gramática con las precedencias implementadas, así como la documentación basada en visualizaciones. El peso en la nota extra de prácticas era de un 20%.

A continuación pasamos a describir las prácticas relacionadas con los analizadores ascendentes. Tres de ellas básicas, una de simulación y dos de generadores, y otra aplicativa. La práctica básica de simulación trata sobre *analizadores SLR(1) con JFlap*. Al igual que en la práctica básica de simulación LL(1), JFlap permite la exploración activa del proceso de construcción de un analizador SLR(1), desde los cálculos de cierres de ítems, hasta la construcción de las tablas del autómata. El estudiante también podrá visualizar el comportamiento del analizador con sus propias cadenas de entrada.

Las dos *prácticas básicas de generación de analizadores ascendentes* usan la herramienta CUP [9]. Con ambas prácticas introducimos a los estudiantes en el uso de herramientas profesionales para la generación de analizadores sintácticos y les mostramos la aplicación de los conceptos teóricos relativos a analizadores ascendentes. Así, la primera práctica básica, *Analizadores LR(1) con CUP*, muestra en el generador CUP los conceptos vistos en la construcción de analizadores LR: conjuntos primeros y siguientes, producción de ítems, generación de estados del autómata LR, producción de las tablas acción e ir-a y finalmente conflictos reducción-reducción y reducción-desplazamiento. La segunda práctica básica, *Herramienta generadora CUP*, trata sobre el enlace entre los analizadores léxico y sintáctico. Los estudiantes aprenden cómo utilizar analizadores léxicos construidos con JFlex, así como usar otras funcionalidades auxiliares que ellos mismos hayan desarrollado e integrarlas en la especificación CUP.

La práctica aplicativa sobre generadores de analizadores ascendentes, *Recuperación de errores con CUP*, como su propio nombre indica trata sobre la recuperación de errores sintácticos. CUP utiliza una estrategia de recuperación en modo pánico similar a YACC, mediante la inserción de puntos de sincronización en la gramática con el símbolo terminal "error". En esta práctica se pide a los estudiantes que diseñen recuperaciones para tres errores concretos en una gramática simple (5 no terminales con 9 producciones) procesando la mayor cantidad posible de la entrada. Y finalmente, en una gramática más compleja (9 no terminales, NT, con 20 reglas) en la que se pide diseñar la mejor recuperación de errores posible usando el menor número de puntos de sincronización. El plazo de entrega fueron cinco días, debiendo proporcionar las especificaciones CUP con las recuperaciones de errores. El peso en la nota extra de prácticas era de un 20%.

En resumen, hemos conseguido que los estudiantes se ejerciten en el diseño de APs y GICs, vean la conexión con dos generadores de analizadores sintácticos y practiquen con ellos antes de afrontar la parte del analizador sintáctico correspondiente a la práctica final.

4.3. Traducción dirigida por la sintaxis

En la fase de traducción dirigida por la sintaxis (TDS) planificamos una práctica aplicativa, que cubre dos conceptos importantes: *TDS con definiciones dirigidas por la sintaxis con CUP* y *TDS con esquemas de traducción con ANTLR*.

En ambas partes se comienza con un enfoque tutorial explicando cómo añadir acciones semánticas a las producciones y cómo usar atributos asociados a los símbolos no terminales. En la parte dedicada a definiciones dirigidas por la sintaxis con CUP se proporciona a los estudiantes los analizadores léxico y sintáctico correspondiente a una gramática simple de cálculo de expresiones lógicas (4 NTs con 10 reglas). En la parte dedicada a esquemas de traducción con ANTLR se proporciona a los estudiantes la especificación ANTLR léxica y sintáctica de una gramática simple de cálculo de expresiones aritméticas (6 NTs con 12 reglas).

En ambos casos la complejidad no se encuentra en el tamaño de la gramática sino en la selección de atributos a utilizar y la inserción de las acciones semánticas pertinentes. También se pidió a los estudiantes que documentaran su práctica con animaciones que mostraran el funcionamiento de sus traductores. Finalmente distribuimos estas animaciones para que, además del profesor de la asignatura, los estudiantes evaluaran de forma anónima las animaciones de otros estudiantes. El plazo de entrega fueron trece días para la entrega de los dos traductores con su documentación y otros 15 para la evaluación de las animaciones asignadas, 6 por cada estudiante. El peso en la nota extra de prácticas era de un 35%.

En resumen, hemos conseguido que los estudiantes se ejerciten en el diseño de traductores dirigidos por la sintaxis con ANTLR y con CUP, y que además hagan un esfuerzo de reflexión sobre las soluciones aportadas al evaluar la documentación de otros estudiantes. Esto les ayudará a la hora de afrontar la parte de traducción dirigida por la sintaxis correspondiente a la práctica final.

4.4. Análisis semántico

En la fase de análisis semántico planificamos una práctica aplicativa sobre *comprobación de tipos y manejo de tabla de símbolos*.

En esta práctica se pide a los estudiantes que diseñen una acción semántica asociada a una producción que representa una llamada a un método de una clase.

En concreto, los estudiantes deben completar la acción semántica dada para que compruebe que la llamada al método efectuada es correcta. También deben implementar dentro de dicha acción la propagación del resultado de la acción (devolución de tipo), aportando además un mensaje concreto para cada tipo de error que pudiera darse. También deben aportar la infertaz de todas las clases adicionales que se necesiten (como por ejemplo, la que implemente la tabla de símbolos). Estas interfaces deben contener para cada método un detallado comentario que indique la funcionalidad del método, qué entrada recibe y qué salida devuelve.

El plazo de entrega fueron siete días y el peso en la nota extra de prácticas era de un 15%.

4.5. Distribución temporal de las prácticas

La planificación de estas prácticas se sincronizó en la medida de lo posible con los contenidos de las clases de teoría. Así las prácticas básicas de simulación estaban perfectamente sincronizadas con las sesiones de teoría donde se explicaron los algoritmos correspondientes.

En la Tabla 1 se puede ver la planificación temporal de las prácticas básicas y aplicativas. Los huecos existentes en las semanas corresponden a sesiones de prácticas dedicadas a la práctica obligatoria.

Se puede observar que las primeras 16 semanas, el primer cuatrimestre, tiene muchas más prácticas que el segundo. Esto se debe a la necesidad de que los alumnos lleguen al segundo cuatrimestre con los conceptos relativos al análisis sintáctico bien asentados.

Las dos primeras entregas de la práctica final consisten en generar especificaciones relativamente sencillas del analizador léxico y sintáctico. Sin embargo, la tercera entrega consistía en el analizador semántico y generador de código. Esta entrega es bastante más compleja y requiere de un esfuerzo de diseño y programación mucho mayor. Por ello decidimos liberar el segundo cuatrimestre de prácticas básicas centrándonos en las aplicativas.

Semana	Práctica	Tipo
2	AFDs y Expresiones reg. JFlap	B
3	Herramienta JFlex	B
4	Traductores con JFlex	**A**
4	APs y GICs con JFlap	B
8	Analizadores LL(1) con JFLap	B
9	Herramienta ANTLR	B
10	Analizadores LL(1) con ANTLR	**A**
11	Analizadores SLR(1) con JFLap	B
12	Analizadores LR(1) con CUP	B
13	Herramienta CUP	B
14-15	Recuperación de errores sintácticos en CUP	**A**
17-20	TDS con ANTLR y CUP	**A**
23	Análisis semántico	**A**

Tabla 1. Distribución temporal de las prácticas básicas (B) y aplicativas (A).

5. Resultados de los estudiantes

Analizamos los resultados de los estudiantes tras utilizar este enfoque de práctica final a lo largo del curso junto con prácticas pequeñas. Como se puede ver en la Figura 3, los resultados han mejorado significativamente. Más del 83% de los grupos registrados entregaron con éxito el analizador léxico y ninguno abandonó en el sintáctico. La entrega final de junio sí sufrió un poco de abandono, pero al final del curso más de un 86% de los grupos registrados superaron la práctica final con éxito.

En cuanto al porcentaje de estudiantes que han aprobado la práctica, representa un 72,27% de los matriculados.

6. Conclusiones

Hemos presentado nuestra experiencia con la asignatura de procesadores del lenguaje. Uno de los puntos importantes son las prácticas. Elegimos la práctica larga porque permite a los estudiantes afrontar una problema más realista y tomar conciencia de las relaciones existentes entre las distintas fases de análisis y síntesis.

Al principio los resultados no fueron buenos. Tratamos de mejorarlos modificando ciertos aspectos burocráticos de la práctica. Esto ayudó a mejorar la tasa de abandono en casi un 22%. Aún así, en junio los grupos no pasaban del 38,22% y en septiembre del 58,7%.

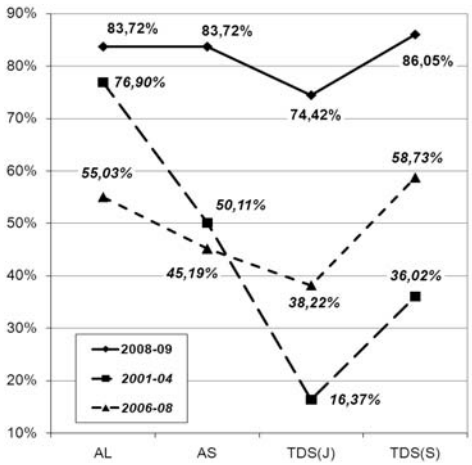

Figura 3. Gráfica del año 2008-09. Los datos de los dos periodos anteriores, 2001-04 y 2006-08, se han resumido con las medias de cada entrega

Llegamos a la conclusión de que la prácticaúnica puede desconectar la teoría y la práctica en esta asignatura. Por ello la reforzamos con prácticas pequeñas de dos tipos, las que trabajan los conceptos teóricos y las que enganchan la teoría con los generadores de analizadores.

El efecto ha sido claramente positivo. Hemos conseguido anular la tasa de abandono entre el léxico y el sintáctico, en junio más de un 74% de grupos registrados completaron la práctica con éxito y hasta septiembre más de un 86%. Lo que en términos de estudiantes, significa un 72% de los matriculados.

Agradecimientos

Este trabajo se ha financiado con el proyecto TIN2008-04103 del Ministerio de Ciencia y Tecnología del Reino de España.

Referencias

[1] Aho, A.V. *Teaching the compilers course*. SIGCSE Bulletin 40, 4 (Nov. 2008), 6-8.

[2] Almeida-Martínez, F.J., Urquiza-Fuentes J. y Velázquez-Iturbide, J.A. *Visualization of Syntax Trees for Language Processing Courses*. Journal of Universal Computer Science, 15(7) (Abr. 2009), 1546-1561.

[3] Aycock, J. *The ART of compiler construction projects*. SIGPLAN Not. 38, 12 (Dec. 2003), 28-32.

[4] Baldwin, D. *A compiler for teaching about compilers*. SIGCSE Bull. 35, 1 (Jan. 2003), 220-223.

[5] Bovet, J. *ANTLRWorks: The ANTLR GUI Development_Environment* http://www.antlr.org/works/ (2010)

[6] Debray, S. *Making compiler design relevant for students who will (most likely) never design a compiler*. SIGCSE Bull. 34, 1 (Mar. 2002), 341-345.

[7] Demaille, A. *Making compiler construction projects relevant to core curriculums*. SIGCSE Bull. 37, 3 (Sep. 2005), 266-270.

[8] Henry, T. *Teaching compiler construction using a domain specific language*. SIGCSE Bull. 37, 1 (Feb. 2005), 7-11.

[9] Hudson, S. *LALR Parser Generator in Java* http://www.cs.tum.edu/projects/cup/ (2010)

[10] Klein, G., Rowe, S., y Décamps, R. *JFlex - The Fast Scanner Generator for Java.* http://jflex.de/ (2010)

[11] Parr, T. *ANTLR Parser Generator* http://www.antlr.org/ (2010)

[12] Rodger, S.H. *Learning automata and formal languages interactively with JFLAP*. SIGCSE Bull. 38, 3 (Sep. 2006), 360.

[13] Ruckert, M. *Teaching compiler construction and language design: making the case for unusual compiler projects with postscript as the target language*. SIGCSE Bull. 39, 1 (Mar. 2007), 435-439.

[14] Sathi, H.L. *A project-based course in compiler construction*. SIGCSE Bull. 18, 1 (Feb. 1986), 114-119.

[15] Shapiro, H.D. y Mickunas, M. D. *A new approach to teaching a first course in compiler construction*. SIGCSE Bull. 8, 1 (Feb. 1976), 158-166.

[16] Waite, W.M. *The compiler course in today's curriculum: three strategies*. SIGCSE Bull. 38, 1 (Mar. 2006), 87-91.

[17] White, E., Sen, R., y Stewart, N. *Hide and show: using real compiler for teaching*. SIGCSE Bull. 37, 1 (Feb. 2005), 12-16.

[18] Xu, L. and Martin, F.G. *Chirp on crickets: teaching compilers using an embedded robot controller*. SIGCSE Bull. 38, 1 (Mar. 2006), 82-86.