Utilización de Técnicas de Análisis Automático para la Asistencia en Entornos de Introducción a la Programación

Maximiliano Chaves[1], María Marta Novaira[1],
Sonia Permigiani[1], and Nazareno Aguirre[1,2]

[1] Departamento de Computación, FCEFQyN, Universidad Nacional de Río Cuarto, Río Cuarto, Córdoba, Argentina
{mchaves, mnovaira, spermigiani, naguirre}@dc.exa.unrc.edu.ar
[2] Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET), Argentina

Resumen Durante el proceso de aprendizaje de conceptos elementales de programación, los estudiantes cometen errores que derivan en programas incorrectos. Las características y tipos de errores cometidos son influenciados por diversos factores, desde el lenguaje de programación y paradigma utilizados, hasta el tipo de ejercicios de programación elegidos y la forma metodológica adoptada para la enseñanza de la programación, entre otros. En este trabajo realizamos una evaluación del tipo de errores cometidos por estudiantes de una asignatura introductoria a la programación, en el primer año de las carreras de Computación de la Universidad Nacional de Río Cuarto. Además de realizar un análisis del tipo de errores más frecuentes, y los conceptos que resultan más difíciles para los estudiantes, estudiamos cómo se podría aprovechar técnicas de análisis automático para desarrollar una herramienta de asistencia a la programación, para entornos introductorios. Analizamos el tipo de herramientas de análisis automático que pueden utilizarse para dar apoyo a los estudiantes, evaluamos su alcance y poder en un conjunto de ejercicios de programación reales, y damos algunos de los lineamientos que guiarán el desarrollo de un entorno de asistencia a la programación, que incorpore estas técnicas.

1. Introducción

La programación es una actividad intelectualmente desafiante, y su aprendizaje, como en otros campos con estas características, es complejo, puede ser frustrante y desalentador. Esto trae aparejado que la asistencia presencial, a través de docentes que acompañen el aprendizaje, asistan con las tareas en prácticas y laboratorios, puede influenciar significativamente la efectividad en el aprendizaje de los estudiantes [2]. Sin embargo, resolver problemas algorítmicos es en muchos casos una tarea que demanda tiempo, y que para muchos estudiantes es difícil de completar enteramente en el marco de actividades supervisadas, como clases prácticas y laboratorios. Más aún, en general los recursos humanos destinados a asignaturas introductorias de programación son limitados, y, especialmente en

cursos numerosos, la relación entre número de docentes y de alumnos dificulta aún más la posibilidad de asistir adecuadamente a muchos estudiantes [2]. Estas observaciones, acompañadas del creciente interés en formar estudiantes con las bases de la programación, y los avances tecnológicos y de comunicaciones actuales, han generado que se considere seriamente la posibilidad de construir asistentes automáticos para la programación. En efecto, si bien algunos de los antecedentes en esta dirección son bastante previos [3], las mencionadas innovaciones tecnológicas han reflotado el interés en este tipo de herramientas de asistencia [11].

Las características y tipos de errores cometidos por estudiantes en contextos introductorios a la programación son variados, y son influenciados por diversos factores, desde el lenguaje de programación y paradigma utilizados, hasta el tipo de ejercicios de programación elegidos y la forma metodológica adoptada para la enseñanza de la programación, entre otros [5,4]. En este trabajo realizamos una evaluación del tipo de errores cometidos por estudiantes de una asignatura introductoria a la programación, en el primer año de las carreras de Computación de la Universidad Nacional de Río Cuarto. La fuente utilizada para el análisis proviene de ejercicios y resoluciones correspondientes a exámenes de varias ediciones de la materia, todas de similares características: la materia se basa en la adopción del paradigma imperativo, utiliza el lenguaje de programación Pascal, pone énfasis en programación estructurada y el uso de diversas abstracciones procedimentales, e incluye un uso básico de especificaciones. En base al análisis del tipo de errores más frecuentes, y los conceptos que resultan más difíciles para los estudiantes, estudiamos cómo podríamos aprovechar los mismos para desarrollar una herramienta de asistencia a la programación, para entornos introductorios. En particular, evaluamos si algunas herramientas de análisis automático modernas tienen el potencial de asistir a los estudiantes en resolver los errores identificados. El objetivo final será incorporar estas técnicas de análisis en una herramienta de asistencia a la programación para entornos introductorios.

2. Análisis de Errores y sus Categorías

Los errores que cometen los estudiantes en cursos introductorios de programación son muy variados, pero pueden ser agrupados en categorías. En este trabajo tomamos como punto de partida dos categorías previamente propuestas, una más general sobre programación [4], y otra más específica para el lenguaje Java [5]. Debido a la motivación de este trabajo, descartamos algunas categorías de errores, que en general son identificados y/o capturados por un compilador estándar (y por lo tanto, ya cuentan con soporte automático para asistir al estudiante). Un ejemplo claro de este tipo de errores es la confusión entre el operador de igualdad y la asignación, que prácticamente cualquier compilador moderno ayuda a detectar. Otras categorías de errores que decidimos ignorar son aquellos que no se aplican al lenguaje Pascal, tales como errores de clases y objetos, de interfaces visuales o de formato en la entrada/salida. Finalmente, también

dejamos fuera del estudio errores correspondientes a temas más avanzados, y que exceden los vistos en cursos introductorios, como errores en implementaciones de tipos abstractos de datos, excepciones, etc. Las categorías resultantes se describen a continuación.

Resolución de Problemas. Esta categoría describe los errores que devienen de la dificultad de un estudiante de comprender un problema y/o cómo implementar una solución. Esto resulta en soluciones parciales de un algoritmo, o desarrollar algo completamente diferente a lo pedido.

Punteros. Esta categoía corresponde a errores con manejo de punteros, como por ejemplo en la implementación de estructuras enlazadas. Dentro de esta categoría se destacan el acceso incorrecto a punteros, la pérdida de referencias, y la liberación incorrecta de memoria.

Ciclos. Corresponde a errores en el uso de estructuras cíclicas del lenguaje. Incluye ciclos infinitos, y errores de tipo "off by one".

Arreglos. Esta categoría incluye los errores específicos del uso de arreglos, como errores "off by one" y acceso fuera de límites.

Condicionales. Corresponde a los errores en el uso de estructuras condicionales, como lógica incorrecta en las condiciones, o casos Else faltantes o incorrectos en los bloques Case.

Parámetros. Esta categoría incluye errores de pasaje de parámetros, tales como pasajes por referencia innecesarios, parámetros pasados por valor que necesitan ser modificados, funciones con parámetros pasados por referencia, etc.

Sintácticos. Esta categoría agrupa errores en el uso de Pascal, como el uso incorrecto de punto y coma, posicionamiento inadecuado de Begin y End, definiciones de tipos incorrectas, errores en la escritura de estructuras cíclicas o condicionales, etc.

Retornos. Los errores de retornos incluyen: funciones que no retornan valores para todas las ramas, uso de salida estándar dentro de un módulo con variable de retorno o funciones, uso de variables globales para guardar información de retorno, etc.

2.1. Proceso de Captura

Luego de la identificación de las categorías relevantes de errores, realizamos una investigación que consistió en identificar cuáles son los errores de programación más comunes en los estudiantes de la asignatura introductoria a la programación de las carreras de Computación de la Universidad Nacional de Río

Cuarto. El análisis se basó en las categorías definidas anteriormente, y en la realización de una evaluación cuantitativa de cantidad de errores en cada categoría.

Para el análisis se tomaron exámenes parciales de la asignatura *Introducción a la Algorítmica y Programación* de las carreras de Computación de la Universidad Nacional de Río Cuarto. Se seleccionaron exámenes de los años 2011 y 2012, de temáticas introductorias a la programación tales como estructuras condicionales, estructuras cíclicas, abstracciones funcionales y procedimentales, tipo de pasaje de parámetros, implementación de estructuras de datos usando memoria dinámica (con punteros), uso de arreglos, etc. Cabe mencionar que la materia utiliza un pseudo-código para el diseño de soluciones algorítmicas, que luego son implementadas en Pascal.

2.2. Análisis de Datos

El análisis de los errores de los exámenes se presenta en la Tabla 1. Los errores de Parámetros son los más comunes, un 22 % de los exámenes presentaban este tipo de errores. Los problemas más comunes en esta categoría son errores de pasajes de parámetro, esto es, modificar una variable o estructura que fue pasada por valor, o problemas de malas prácticas en pasar parámetros por referencia innecesariamente. En segundo lugar están los errores de Resolución de Problemas. Los problemas en esta categoría refieren a ejercicios en que los alumnos no entendieron la consigna del problema a resolver, o que entendido los requisitos la solución, no resuelven correctamente lo planteado. El tercer lugar es compartido por dos tipos de errores, Punteros y Retornos. En el primero se encuentran errores de tratamiento de secuencias simplemente encadenadas, como actualizaciones erróneas de referencias que derivan en la pérdida de propiedades de la secuencia. En el caso de errores de retorno, éstos significan una falta de comprensión por parte del estudiante del concepto de retorno de un valor en el caso de funciones, y el manejo de variables de resultado en el caso de una acción. Esto resulta en errores de retornos dentro de solo una rama de una condición, funciones con parámetros que guardan resultados, funciones que escriben por pantalla (salida estándar) resultados; y procedimientos que manejan variables globales para los resultados, o que también escriben los resultados por pantalla.

Daremos dos ejemplos específicos de las categorías de errores más comunes. Más ejemplos pueden encontrarse en el reporte técnico disponible en `http://dc.exa.unrc.edu.ar/staff/naguirre/education/`.

Ejercicio 1 (Segundo Parcial 2011) Dada una matriz cargada, de n filas y m columnas, desarrollar una función que cumpla con la siguiente especificación:

$$PRE : a = [a_{(1,1)}, a_{(1,2)}, ..., a_{(n,m)}] \wedge 1 \leq n \wedge 1 \leq m \wedge num = num_0|$$

$$DEF : esConstante(a, num) = \exists c(1 \leq c \leq m) : \forall f(1 \leq f \leq m) : a[f, c] = num$$

Errores	2011	Porcentaje	2012	Porcentaje	Total	Porcentaje Total
Parámetros	2	24 %	13	15 %	77	22 %
Resolución de problemas	56	21 %	10	12 %	66	18 %
Punteros	26	9 %	21	25 %	47	13 %
Retornos	2	15 %	5	6 %	46	13 %
Arreglos	2	9 %	5	6 %	30	8 %
Condicionales	2	7 %	9	10 %	29	8 %
Sintácticos	2	9 %	5	6 %	31	8 %
Ciclos	2	3 %	15	18 %	23	6 %

Cuadro 1. Tabla de errores más comunes

La siguiente resolución no resuelve correctamente el problema. El ciclo no termina cuando se encuentra el elemento; no hay retorno en el caso de no haber elemento que cumpla con el criterio de búsqueda.

```
Function esConstante (a : TArray; num:Integer): boolean;
Var
    i,j : Integer;
Begin
    p := True;
    i := 1;
    For j := 1 To Column Do
        While p = True And i<> Row Do
            If a[i,j] = num Then
                p:= True
            Else
                p:= False
            i := i + 1;
        End;
        If p = True Then
            ReturnesConstante:= True
        Else
            p := True
        End;
    End;
End;
```

Ejercicio 2 (Segundo parcial 2012) Dado un arreglo de 120 valores lógicos, determinar si la cantidad de valores Verdaderos es mayor a la cantidad de Falsos, en la primera mitad del arreglo. Resolver mediante una función.

La siguiente resolución tiene un error de retorno; falta indicar qué se retorna en una rama de la condición.

```
...
Begin
    i:= 1;
    While (i<= 60) Do
        If arre[i] = True
        then
            v := v+1
        else
            f := f+1
        i := i+1;
    end;
    if v > f
    Then
        CountVF := v
end;
```

3. Herramientas de Análisis

En base al análisis de los errores más comunes de programación presentados en la sección anterior, se derivan dos tipos de herramientas de asistencia para su solución. Por un lado, herramientas de análisis de código que puedan detectar errores que exceden aquellos que un proceso de compilación estándar normalmente identifica. En este tipo entran todas las categorías de errores a excepción de los errores de Resolución de Problemas. Asistencias relativas a este tipo de errores ayudarían, por ejemplo, a corregir y sugerir soluciones a formar de pasajes de parámetros erróneas, sugerencias sobre el uso de entrada y salida en módulos, detección de código duplicado, asistencia en la construcción de expresiones lógicas, etc.

En el otro tipo de herramienta de análisis debería apuntar a asistir en los errores que surgen de resolución de problemas. Este tipo de errores es sustancialmente más difícil de detectar, dado que esencialmente corresponden a detectar *diferencias semánticas*, es decir, discrepancias entre el comportamiento esperado de un programa y el comportamiento real del mismo. Es sabido que detectar tales problemas requiere de una especificación formal del comportamiento esperado del programa, y el problema es indecidible [6], pues corresponde esencialmente a comprobar que un programa satisface su especificación. Existen sin embargo, como mostraremos más abajo, formas de comprobar, al menos parcialmente, que un programa cumple con el comportamiento esperado del mismo.

3.1. Errores de Parámetros: Pascal Analyzer

Para la resolución de errores de tipos de pasajes de parámetros y similares, una técnica utilizada con frecuencia es el análisis estático extendido [7]. Como en el caso particular de nuestro entorno de evaluación el lenguaje utilizado es

Pascal, necesitamos una herramienta de este tipo para este lenguaje. Lamentablemente, estas herramientas no abundan para este lenguaje. El único ejemplo que conseguimos encontrar es una herramienta comercial denominada Pascal Analyzer (PAL). Esta herramienta realiza un análisis estático extendido y genera un reporte en base a una implementación en Delphi (Pascal orientado a objetos). La herramienta recibe un conjunto de archivos fuente, y presenta en distintas categorías, informes, correcciones, sugerencias y advertencias sobre el código, generados en base a un análisis estático extendido.

Esta herramienta produce salidas sumamente detalladas para una cantidad de errores posibles, muchos de los cuales sólo se aplican a programas Delphi. En nuestro caso, sólo estamos interesados, en principio, en los errores que tienen que ver con pasaje de parámetros. La herramienta es razonablemente potente en este sentido, permitiendo detectar errores tales como variables pasadas por referencia y no seteadas, variables pasadas por valor y modificadas en el cuerpo de la rutina, entre muchos otros.

3.2. Errores de Resolución de Problemas: Pex

Como mencionamos anteriormente, los errores en resolución de problemas son sustancialmente más difíciles, en relación a la provisión de asistencia automática al estudiante. En este aspecto, nuestro candidato inicial, como tecnología de análisis automático a utilizar, es la ejecución concreta/simbólica [10]. Específicamente, proponemos utilizar Pex [9], una herramienta de generación automática de tests, en un sentido similar al propuesto en [11]. Pex genera tests automáticamente intentando cubrir ramas en el código. La idea esencial consiste en utilizar una implementación secreta, considerada correcta, y provista por el docente. Luego, se utilizarUa Pex para intentar cubrir ramas en el siguiente programa:

```
program (x) {
    if (implementacionEstudiante(x) != implementacionSecreta(x))
        throw new Exception();
}
```

Pex intentará activamente, mediante constraint solving, cubrir ramas de este programa, en particular la rama en la cual se lanza una excepción. Notemos que si tal rama es cubierta, lo que Pex proveerá es valores para los parámetros del programa para los cuales los resultados obtenidos por la solución del estudiante y la implementación secreta del docente, difieren.

Una alternativa al uso de Pex sería simplemente contar con una cantidad de tests provistos por el docente, que capturen, para una cantidad limitada de escenarios particulares, el comportamiento esperado del programa a desarrollar. La ventaja de utilizar una herramienta como Pex es que el análisis es sustancialmente más exhaustivo que el que, en principio, se puede realizar sólo mediante el uso de una cantidad acotada, ad hoc, de tests.

En el marco de este trabajo, el uso de Pex como herramienta de asistencia no es directo. La herramienta funciona para proyectos en cualquier lenguaje de programación sobre .NET, por ejemplo, C#. No soporta Pascal, lamentablemente.

Esto nos obliga entonces al uso de traductores de código Pascal a C# (u otro lenguaje .NET). Esta traducción no es simple, y dificulta la aplicación del análisis en muchos casos. Por ejemplo, la traducción de un programa Pascal a C# hace que el programa C# resultante, al ser compilado, detecte muchos problemas (no de tipo semántico) que el compilador Pascal pasa por alto. Concretamente, el compilador Pascal compila normalmente si un algoritmo tiene una función que no retorna en todas las ramas, mientras que el compilador de C# sí. Por otra parte, el funcionamiento de los dos lenguajes involucrados en la traducción tienen semánticas, en particular en lo referido a pasaje de parámetros, muy diferentes. En particular, módulos con parámetros pasados por referencia en Pascal no son fáciles de traducir a C#, donde el único tipo de pasaje de parámetros es por valor.

3.3. Evaluación de las Técnicas de Análisis

Nuestra evaluación de las técnicas de análisis elegidas se realizó sobre un amplio número de casos de estudio, tomados de resoluciones reales de exámenes parciales. En esta sección, y por razones de espacio, sólo mostramos los resultados obtenidos por parte de las herramientas antes descriptas, para los dos ejemplos mencionados anteriormente en el artículo. Más datos pueden encontrarse en el reporte técnico extendido correspondiente a este artículo.

Ejercicio 1. Dado el tipo de error en la resolución ejemplo de este ejercicio, la herramienta a utilizar aquí es Pex. Al proveer a Pex de una versión correcta del programa (secreta para el estudiante) y contrastarla como hemos indicado anteriormente, con la resolución del estudiante, obtenemos el siguiente contra-ejemplo:

```
a = {{0,1},{0,0}}; num = 1
```

En este caso el programa del estudiante retorna true (debido a que el valor de la variable i no es reseteado luego de la ejecución del while), mientras que el programa secreto (correcto) retorna false. Este es sólo un ejemplo de un error detectado por la diferencia semántica realizada por Pex. Otros inputs reportados por la herramienta detectan problemas de terminación en el programa del estudiante.

Ejercicio 2. Dado el tipo de error en la resolución ejemplo de este ejercicio, la herramienta a utilizar aquí es Pascal Analyzer. Pascal Analyzer produce, entre el enorme reporte de problemas para este ejercicio, lo siguiente:

```
Local variables that are referenced before they are set:
f : Integer                    Var, Local
```

Efectivamente, el analizador estático extendido detecta en este caso que existen caminos de ejecución del programa en los cuales se referencia (hace uso) de f, antes de ser seteada (en un momento en el cual contiene basura).

Es interesante mencionar además que otros errores de tipo de manejo de parámetros y entrada/salida, como en este ejemplo el de no tener un valor de retorno en cada rama, es detectado al intentar utilizar Pex, durante la traducción y compilación en C#.

4. Conclusiones y Trabajos Futuros

En este artículo hemos analizado cuantitativamente los errores que más afectan a estudiantes de las carreras de Computación de la Universidad Nacional de Río Cuarto, en la asignatura introductoria a la programación. La finalidad de este estudio es, como explicamos en el artículo, evaluar si algunas técnicas de análisis automático modernas podrían aplicarse para dar asistencia automática a los estudiantes, y así contribuir a lidiar con algunos problemas comunes de aprendizaje en los estudiantes. Identificamos los errores más comunes, y analizamos para dos de ellos la utilización de análisis estático extendido, y de ejecución concreta/simbólica, como tecnologías de análisi que podrían aprovecharse para la asistencia. Elegimos herramientas concretas de análisis y las aplicamos a un número de ejemplos concretos, tomados de resoluciones reales de exámenes parciales. A pesar de las dificultades que tienen que ver con el lenguaje de programación usado en la asignatura (Pascal) y la casi total ausencia de herramientas de análisis para este lenguaje, descubrimos que muchos de los errores cometidos por los estudiantes pueden ser identificados a través del uso de las técnicas mencionadas, satisfactoriamente. Sin embargo, la forma en que estos errores son reportados es en general bastante críptica y técnica, y por lo tanto se debería post-procesar la salida de estas herramientas para dar feedback a los estudiantes de manera más comprensible y dirigida.

En cuanto a trabajos futuros, como hemos mencionado en diversas ocasiones en el artículo, nuestra finalidad es desarrollar una herramienta de asistencia a los estudiantes. En base al estudio realizado y a la evaluación preliminar de las técnicas de análisis presentada, comenzaremos a implementar un primer prototipo de asistente. Planeamos poder evaluar la adopción del asistente, como así también en valor del feedback provisto por los estudiantes, en un estudio futuro. También extenderemos el análisis presentado en este artículo a otras categorías de errores, que seguramente demandarán incorporar otras tecnologías de análisis, además de las ya analizadas.

Finalmente, es parte de nuestra intención desarrollar a futuro esquemas de gamification [8] que faciliten la adopción de la plataforma de asistencia automática a la programación, siguiendo el relativo éxito que ha tenido esta política en contextos similares al nuestro [1].

Referencias

1. J. Bishop, R. Horspool, T. Xie, N. Tillmann y J. de Halleux, *Code hunt: experience with coding contests at scale*, Proceedings of the 37th International Conference on Software Engineering ICSE 2015, IEEE Press, 2015.
2. B. Bloom, *The 2 Sigma Problem: The Search for Methods of Group Instruction as Effective as One-to-One Tutoring*, Educational Researcher 13(6), 1984.
3. R. Farrell, J. Anderson y B. Reiser, *An Interactive Computer-Based Tutor for LISP*, Proceedings of the National Conference on Artificial Intelligence, AAAI Press, 1984.
4. A. Hansen, *An Empirical Study of Students Programing Bugs*, Utah State University, Logan, UT, 2011.
5. M. Hristova, A. Misra, M. Rutter y R. Mercuri, *Identifying and Correcting Java Programming Errors for Introductory Computer Science Students*, en Proceedings de 34th SIGCSE Technical Symposium on Computer Science Education SICGSE 2003, ACM, 2003.
6. C. Nelson, *Techniques for program verification*, Doctoral Dissertation, Stanford University, 1980.
7. F. Nielson, H, Nielson y C. Hankin, *Principles of Program Analysis*, Springer, 2010.
8. K. Seaborn y D. Fels, *Gamification in theory and action*, International Journal of Human-Computer Studies 74(C), Academic Press, 2015.
9. N. Tillmann y J. de Halleux, *Pex: white box test generation for .NET*, Proceedings of the 2nd international conference on Tests and proofs TAP 2008, LNCS, Springer, 2008.
10. N. Williams, B. Marre, P. Mouy, *On-the-Fly Generation of K-Path Tests for C Functions*, Proceedings of the 19th IEEE International Conference on Automated Software Engineering (ASE 2004), IEEE Computer Society, 2004.
11. T. Xie, N. Tillmann y J. de Halleux, *Educational software engineering: where software engineering, education, and gaming meet*, Proceedings of the 3rd International Workshop on Games and Software Engineering GAS 2013, IEEE Press, 2013.