Composición Categórica de Análisis Automáticos para Líneas de Productos Extendidas

Daniel-Jesus Munoz, Monica Pinto y Lidia Fuentes

ITIS Software, CAOSD, Universidad de Málaga, Andalucía Tech, España
{danimg,pinto,lff}@lcc.uma.es,
http://caosd.lcc.uma.es

Resumen Para las *Líneas de Productos Software* (LPS) se necesitan operaciones que nos permitan analizar dicho software y el reúso de sus características. Los razonadores son herramientas que automatizan estas operaciones. Desde la extensión de LPS con diversos tipos atributos de calidad, el tipo y número de operaciones de razonamiento ha crecido más rápido que el desarrollo de los respectivos razonadores. En consecuencia, las operaciones de análisis extendido son en el mejor caso *parcialmente* soportadas por los razonadores estado-del-arte.
Para este desafío, podemos aplicar un enfoque de *Teoría de Categorías* (TC); el álgebra abstracta que capta los componentes comunes de estructuras aparentemente diferentes. Basándonos en la flexibilidad de sus razonamientos, proponemos una metodología donde las operaciones extendidas sean composiciones configurables de un conjunto de operaciones reusables independientes. Por tanto, buscamos definir e implementar un framework de razonamiento funcional de LPS extendidas basado en TC.

Keywords: teoría de categorías · modelo de variabilidad · consultas · razonador · atributo de calidad

1. Introducción

Una *Línea de Productos Software* (LPS) es un conjunto de software y sistemas que comparten una serie de **características** reusables prescritas [5]. Debido a la escasez de recursos, se ha potenciado el desarrollo de sistemas óptimos y eficientes (e.g., reducir tiempo de ejecución). Consecuentemente, la representación de LPS en modelos se ha extendido con *Atributos de Calidad* (AC) tanto a nivel de características junto sus respectivas funciones agregadoras (e.g., suma de coste) [9], como a nivel de producto (e.g., consumo energético) [6]. De ahora en adelante, nos referiremos genéricamente a su representación como *Modelos de Variabilidad con Calidad* (MVC) [7].

Para automatizar los análisis de las LPS se han desarrollado las herramientas denominadas razonadores [4]. Pero con la llegada de los MVC, el tipo y número de operaciones de razonamiento necesarias ha crecido más rápido que el el estado-del-arte de los razonadores. Esto ocurre por diversas causas, como la incompatibilidad con cualquier tipo de extensión (e.g., razonadores SAT de

fórmulas proposicionales [2]), la complejidad técnica (e.g., restricciones de AC agregados), o el simple abandono por ser soluciones ad hoc que ya cumplieron su propósito [4]. La principal consecuencia es que los razonadores estado-del-arte re-implementan una y otra vez las mismas operaciones básicas.

Para este tipo de desafíos, surgió en los '70s la *Teoría de Categorías* (TC). TC es una teoría matemática general sobre estructuras algebraicas que permite captar y relacionar los aspectos comunes de diferentes estructuras, abstrayéndose de sus particularidades [1]. Debido a la naturaleza abstracta de las categorías, encontramos que los razonadores en TC son altamente flexibles y configurables a través de programación funcional [3]. Entre otras propiedades algebraicas, TC soporta por definición la composición directa de funciones entre distintas categorías, es decir, composición de operaciones. La necesidad, complejidad y potenciales beneficios de la composición de *consultas* a LPS se consideró en [10].

En este trabajo **proponemos** un framework categórico para operaciones de razonamiento avanzado de LPS, las cuales se formen como composiciones configurables de un conjunto de operaciones reusables e independientes. Las **contribuciones** potenciales son (1) definición funcional de operaciones reusables de razonamiento considerando toda la literatura de LPS extendidas, (2) definición de operaciones avanzadas mediante la composición y configuración modular de operaciones, y (3) desarrollo de una interfaz de razonamiento flexible, configurable y directamente extensible en un razonamiento categórico.

2. Composición Categórica de Consultas Configurables

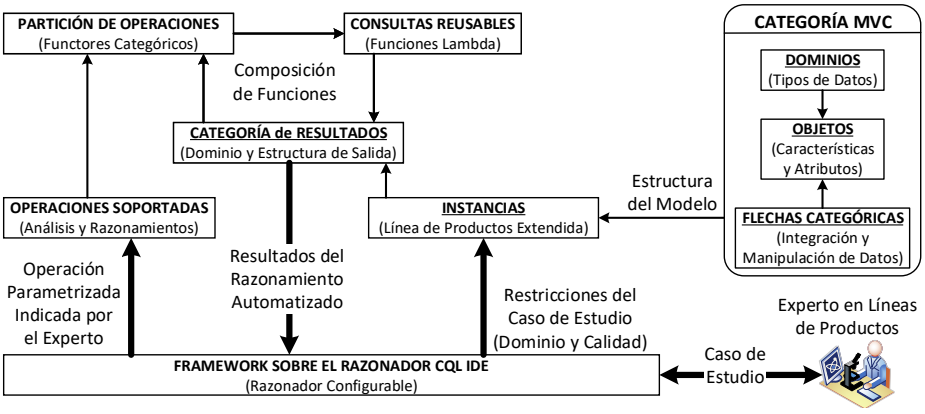

Figura 1. Framework Categórico para un Razonamiento Modular en CQL IDE

En la Figura 1 presentamos nuestra propuesta de framework de TC para un razonamiento modular opaco. Como podemos visualizar en la caja inferior, usamos de base el entorno de desarrollo integrado para TC *Categorical Query Language* (CQL) IDE [8], un software funcional canónico, y que integra razonadores configurables de diferente dominio y que trabajan de manera **concurrente** y **eficiente**: un demostrador automático de teoremas basado en el algoritmo de finalización de Knuth-Bendix para lógica de orden superior y ecuaciones

aritméticas, y hashing, árboles binarios de búsqueda y algoritmos de persecución de objetivos para relaciones, restricciones y requisitos. CQL IDE nos permite definir mediante programación funcional lambda las operaciones de razonamiento reusables, así como su composición.

En la parte superior-derecha de la Figura 1, encontramos la *Categoría MVC*. Esta representa la estructura del MVC de una LPS extendida siguiendo nuestro enfoque de TC detallado en [7]. Resumidamente, una *Categoría* es cualquier colección de *Objetos* que representan espacios perteneciente a diversos dominios, y que pueden relacionarse entre sí mediante funciones denominadas *Flechas* (es decir, morfismos) que preservan composición asociativa e identidad. En el caso de un MVC, los elementos de un objeto son sus características y AC. Estos elementos tienen un dominio definido, por ejemplo, booleano para características clásicas o numérico para la mayoría de AC. En este caso, las flechas son las relaciones jerárquicas y cruzadas de cualquier árbol de variabilidad, así como las de los ACs en sus dos niveles, incluyendo las funciones lambda agregadoras. Un ejemplo sencillo de función lambda agregadora sería la suma de costes $\lambda(x \in producto.caracteristica.coste : x.suma)$, pero otras agregaciones y funciones lambdas complejas son igualmente definible en CQL IDE. Continuando, la LPS completa la obtenemos en *Instancias*, ya que se le asignan valores a los elementos de la categoría MVC. El ejemplo más simple es asignarle un nombre (i.e., dominio cadena de caracteres) a cada característica del MVC.

A nivel de uso, la composición de razonamientos se ejecutaría de manera opaca al usuario. Continuando con un pseudo-ejemplo, un *Experto en LPS* indicaría a CQL IDE el caso de estudio "10 muestras aleatorias para una LPS con restricción (Característica A y coste agregado mayor de 5€) implica (consumo energético menor que 1 vatio)". Primero, nos fijaríamos en la izquierda de la Figura 1, donde nuestro framework comprobaría que la operación `Muestreo Aleatorio 10` está soportada. En paralelo, trasladaríamos la restricción del experto a *Instancias* como una flecha similar pero que filtre el espacio de productos considerando AC. Siguiendo con la operación, *Partición de Operaciones* dividiría la operación en una composición de dos funciones lambda: `Límite 10` $\circ$ `Aleatorio`. La *Consulta Reusable* `Aleatorio` reorganiza los indicadores de las instancias (sin generarlas), y tras ello, la *Consulta Reusable* `Límite` genera los 10 primeros productos que cumplan con las restricciones indicadas por el experto.

Cabe indicar que estas funciones lambda no son simples flechas categóricas, sino operaciones entre categorías denominadas *Functores*. Esto es debido a que cualquier operación sobre una categoría entrante debe definir una estructura de salida en forma de otra categoría saliente. Un símil en consultar a bases de datos sería definir la estructura de salida en `SELECT` y el razonamiento (i.e., restricciones y operaciones) a partir de `FROM`. Igualmente, si anidamos y componemos consultas debemos asegurar que una categoría de salida es compatible con la estructura de la categoría de entrada de la siguiente operación de la secuencia compuesta. Volviendo al ejemplo, definiríamos los functores como: $F_1 : Cat_{Entrada1} \xrightarrow{Aleatorio} Cat_{Salida1}$ y $F_2 : Cat_{Entrada2} \xrightarrow{Muestreo_{10}} Cat_{Salida2}$. *Categoría de Resultados* se encarga de que en operaciones compuestas, $Cat_{Entrada_1}$

sea MVC, Cat_{Salida_x} sea compatible con $Cat_{Entrada_{(x+1)}}$, y que Cat_{Salida_n} sea la esperada por el experto.

Este enfoque es potencialmente compatible y extensible para todo tipo de operaciones, ya sean de: (1) análisis del propio MVC como contar el número de features y ACs, (2) generación de productos en base al razonamiento de la calidad como las búsquedas de productos óptimos, o (3) refactorización como la especialización del MVC.

3. Conclusiones

En este artículo de prospección evaluamos la necesidad y potenciales ventajas de un framework de composición de razonamientos automatizados, reusables y naturalmente extensibles. Así mismo, mostramos la visión general de su implementación modular en la herramienta funcional de TC denominada CQL IDE.

Agradecimientos

Este trabajo está financiado por el programa de investigación e innovación H2020 de la Unión Europea bajo el acuerdo de subvención DAEMON 101017109, por los proyectos también co-financiados por fondos FEDER LEIA UMA18-FEDERJA-15, MEDEA RTI2018-099213-B-I00 y Rhea P18-FR-1081, y la ayuda PRE2019-087496 del Ministerio de Ciencia e Innovación.

Referencias

1. Barr, M., Wells, C.: Category theory for computing science. Prentice Hall (1990)
2. Batory, D.: Feature models, grammars, and propositional formulas. In: Obbink, H., Pohl, K. (eds.) Software Product Lines. pp. 7–20. Springer, Berlin (2005)
3. Breiner, S., Breiner, S., Pollard, B., Subrahmanian, E.: Workshop on Applied Category Theory: Bridging Theory and Practice. US Department of Commerce, National Institute of Standards and Technology (2020)
4. Horcas, J.M., Pinto, M., Fuentes, L.: Software product line engineering: A practical experience. In: Proceedings of the 23rd International Systems and Software Product Line Conference - Volume A. p. 164–176. SPLC '19, Association for Computing Machinery, New York, NY, USA (2019)
5. Käköla, T., Duenas, J.C.: Software product lines. Springer (2006)
6. Munoz, D.J.: Achieving energy efficiency using a software product line approach. In: Proceedings of the 21st SPLC. p. 131–138. SPLC '17, ACM, NY, USA (2017)
7. Munoz, D.J., Gurov, D., Pinto, M., Fuentes, L.: Category theory framework for variability models with non-functional requirements. In: Advanced Information Systems Engineering. pp. 397–413. Springer, Cham (2021)
8. SCHULTZ, P., WISNESKY, R.: Algebraic data integration. Journal of Functional Programming **27**, e24 (2017)
9. Siegmund, N., Sobernig, S., Apel, S.: Attributed variability models: Outside the comfort zone. In: Proceedings of the 11th Joint Meeting on Foundations of Software Engineering. p. 268–278. ACM, New York, New York, USA (2017)
10. Trinidad, P., Ruiz-Cortés, A., Benavides, D.: Automated Analysis of Stateful Feature Models, pp. 375–380. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)