# Resumen

Producir software eficiente y efectivo es una tarea que parece ser tan difícil ahora como lo era para los primeros ordenadores. Con cada mejora de *hardware* y herramientas de desarrollo (como son compiladores y analizadores), la demanda de producir software más rápido y más complejo ha ido aumentando. Por tanto, todos estos análisis auxiliares ahora son una parte integral del desarrollo de programas complejos.

La fragmentación de programas es una técnica de análisis estático, que da respuesta a «¿Qué partes del programa pueden afectar a esta instrucción?», y otras preguntas similares. Su aplicación principal es la depuración de programas, porque puede acotar la zona de código a la que el programador debe prestar atención mientras busca la causa de un error. También tiene otras muchas aplicaciones, como pueden ser la paralelización y especialización de programas, la comprensión de programas y el mantenimiento. En los últimos años, su uso más común ha sido como preproceso a otros análisis con alto coste computacional, para reducir el tamaño del programa a procesar, y, por tanto, el tiempo de ejecución de estos. La estructura de datos más popular para fragmentar programas es el *system dependence graph* (SDG), un grafo dirigido que representa las instrucciones de un programa como vértices, y sus dependencias como arcos. Los dos tipos principales de dependencias son las de control y las de datos, que encapsulan el flujo de control y datos en todas las ejecuciones posibles de un programa.

El área de lenguajes de programación está en eterno cambio, ya sea por la aparición de nuevos lenguajes o por el lanzamiento de nuevas características en lenguajes existentes, como pueden ser Python, Java, Erlang, Rust o Go. Sin embargo, la fragmentación de programas se definió originalmente para (y está aún enfocada a) el paradigma imperativo. Aun así, hay características populares en lenguajes imperativos, como las *arrays* y las excepciones, que aún no tienen una representación eficiente y/o completa en el SDG. Otros paradigmas, como el funcional o el orientado a objetos, sufren también de un soporte parcial en el SDG.

Esta tesis presenta mejoras para construcciones comunes en la programación moderna, dividiendo contribuciones en las enfocadas a dependencias de control y las enfocadas a datos. Para las primeras, (i) especificamos una nueva representación de instrucciones *catch*, junto a una descripción completa del resto de instrucciones relacionadas con excepciones. También (ii) analizamos las técnicas punteras para saltos incondicionales (p.e., *break* o *return*), y mostramos los riesgos de combinarlas con otras técnicas para objetos, llamadas o excepciones. A continuación, ponemos nuestra mirada en la concurrencia, con una (iii) formalización de un depurador de especificaciones CSP reversible y causal-consistente. En cuanto a las dependencias de datos, se enfocan en técnicas sensibles al contexto (es decir, más precisas en presencia de rutinas y sus llamadas). Exploramos las dependencias de datos generadas en programas concurrentes por memoria compartida, (iv) redefiniendo las dependencias de interferencia para hacerlas sensibles al contexto. A continuación, damos un pequeño rodeo por el campo de la indecidibilidad, en el que (v) demostramos que ciertos tipos de análisis de datos sobre programas con (y sin) estructuras de datos complejas son indecidibles. Finalmente, (vi) ampliamos un trabajo previo sobre la fragmentación de estructuras de datos complejas, combinándolo con la fragmentación tabular, que la hace sensible al contexto.

Además, a lo largo de toda la tesis, se han desarrollado o extendido múltiples librerías de código con las mejoras mencionadas anteriormente. Estas librerías nos han permitido realizar evaluaciones empíricas para algunos de los capítulos, y también han sido publicadas bajo licencias libres, que permiten a otros desarrolladores e investigadores extenderlas y contrastarlas con sus propuestas, respectivamente. Las herramientas resultantes son dos fragmentadores de código para Java y Erlang, y un depurador de CSP reversible y causal-consistente.