# MEMORIA PRÁCTICA TLP 2023-24

Pablo Suárez Peña
Psuarez142@alumno.uned.es
53647180w - Asturias

**1 (2 puntos). Supongamos una implementación de la práctica (usando los mismos tipos de datos aquí presentados) en un lenguaje no declarativo (como Java, Pascal, C...). Comente qué ventajas y qué desventajas tendría frente a la implementación en Haskell. Relacione estas ventajas desde el punto de vista de la eficiencia con respecto a la programación y a la ejecución. ¿Cuál sería el principal punto a favor de la implementación en los lenguajes no declarativos? ¿Y el de la implementación en Haskell? Justifique sus respuestas.**

Una implementación en un lenguaje no declarativo, como Java, Pascal o C, tendría ciertas ventajas y desventajas en comparación con la implementación en Haskell:

Ventajas de la implementación en un lenguaje no declarativo:

Control directo sobre la memoria: En lenguajes como C, el programador tiene un mayor control sobre la gestión de la memoria, lo que puede ser ventajoso en términos de eficiencia en aplicaciones de alto rendimiento.
Eficiencia de ejecución: Los lenguajes no declarativos tienden a ser más eficientes en tiempo de ejecución en comparación con los lenguajes funcionales como Haskell. Esto se debe a que los lenguajes no declarativos están más cerca del hardware y permiten optimizaciones a nivel de máquina.
Mayor acceso a bibliotecas y herramientas de bajo nivel: En entornos donde se requiere interactuar con sistemas de bajo nivel o bibliotecas específicas del sistema operativo, los lenguajes no declarativos suelen tener una ventaja debido a su amplia disponibilidad de bibliotecas y herramientas.
Desventajas de la implementación en un lenguaje no declarativo:

Complejidad del código: Los lenguajes no declarativos suelen requerir más líneas de código para lograr el mismo resultado que en un lenguaje funcional. Esto puede hacer que el código sea más propenso a errores y más difícil de mantener.
Menor expresividad: Los lenguajes no declarativos a menudo requieren que el programador se concentre más en cómo se realiza una tarea en lugar de qué tarea se realiza. Esto puede llevar a una menor expresividad y legibilidad del código.
Manejo manual de la concurrencia y la concurrencia: En lenguajes no declarativos, el manejo de la concurrencia y la concurrencia suele ser más complejo y propenso a errores, ya que el programador tiene que administrar los hilos y los bloqueos manualmente.
Eficiencia desde el punto de vista de la programación y la ejecución:

Programación: En términos de eficiencia de programación, los lenguajes no declarativos podrían tener una ventaja debido a su control directo sobre la memoria y su acceso a bibliotecas de bajo nivel, lo que puede facilitar la implementación de algoritmos optimizados.
Ejecución: En términos de eficiencia de ejecución, los lenguajes no declarativos pueden ser más rápidos debido a su proximidad al hardware y las optimizaciones a nivel de máquina que permiten. Sin embargo, esto puede variar dependiendo del caso específico y de la calidad de la implementación.
Principal punto a favor de la implementación en los lenguajes no declarativos:

El principal punto a favor de la implementación en los lenguajes no declarativos es su eficiencia en tiempo de ejecución y su control directo sobre la memoria y el hardware, lo que los hace ideales para aplicaciones que requieren un alto rendimiento y una gestión eficiente de los recursos.

Principal punto a favor de la implementación en Haskell:

El principal punto a favor de la implementación en Haskell es su simplicidad y expresividad, lo que puede llevar a un código más claro, conciso y fácil de mantener. Además, Haskell ofrece características poderosas como la evaluación perezosa y el sistema de tipos estáticos que pueden mejorar la seguridad y la robustez del código.

**2 (1’5 puntos). Indique, con sus palabras, cómo afecta el predicado predefinico no lógico corte (!) al modelo de computación de Prolog. ¿Cómo se realizaría este efecto en Haskell? ¿Considera que sería necesario el uso del corte en una implementación en Prolog de esta práctica? Justifique sus respuestas.**

El predicado predefinido no lógico corte (!) en Prolog tiene un impacto significativo en el modelo de computación al influir en el comportamiento del backtracking. Cuando se encuentra un corte en una regla, Prolog no continúa buscando soluciones alternativas más allá de ese punto en particular. Esto significa que el corte puede ser utilizado para evitar backtracking innecesario y mejorar la eficiencia del programa al descartar ramas de búsqueda que no son relevantes para el resultado deseado.

En Haskell, el efecto del corte no lógico no se puede replicar directamente, ya que Haskell utiliza un modelo de evaluación perezosa y no tiene un mecanismo equivalente al backtracking de Prolog. En Haskell, el control del flujo se logra mediante patrones de recursión, combinadores de funciones y otras construcciones propias de los lenguajes funcionales.

En cuanto a si sería necesario el uso del corte en una implementación en Prolog de esta práctica, depende del enfoque de diseño y de los requisitos específicos del problema. El corte puede ser útil para evitar soluciones redundantes y mejorar la eficiencia del programa. Sin embargo, su uso debe ser cuidadoso, ya que puede ocultar errores lógicos y hacer que el código sea más difícil de entender y mantener.

En general, en Prolog, el uso del corte puede ser una herramienta poderosa para optimizar el rendimiento de ciertas consultas, pero debe usarse con precaución y entender completamente su impacto en el comportamiento del programa. En muchos casos, una implementación limpia y eficiente puede lograrse sin necesidad de utilizar el corte, aprovechando el poder del unificación y el backtracking de Prolog de manera más natural

**3 (1,5 puntos). Indique qué clases de constructores de tipos (ver capítulo 5 del libro de la asignatura) se han utilizado para definir los tipos de datos presentes en el módulo StackMachine. Justifique sus respuestas.**

En el módulo StackMachine se han utilizado tres clases de constructores de tipos para definir los tipos de datos:

Constructores de tipo de datos algebráicos: Estos se utilizan para definir los tipos BOp, UOp y SynTree. Por ejemplo, en la definición de SynTree, se definen tres constructores de tipo de datos algebráicos: Binary, Unary y Operand. Estos constructores representan las diferentes formas que puede tener un árbol de sintaxis abstracta, es decir, un nodo binario, un nodo unario o un operando.
Constructores de tipo de datos parametrizados: Los tipos de datos SynTree y Stack son ejemplos de esto. En la definición de SynTree, el tipo SynTree es parametrizado por un tipo a, lo que significa que puede contener operandos de cualquier tipo a. En la definición de Stack, Stack es parametrizado por un tipo a, lo que permite que la pila pueda contener elementos de cualquier tipo.
Constructores de tipo de datos de enumeración: Se utilizan para definir los tipos de datos BOp y UOp. Estos tipos enumeran los diferentes operadores binarios (ADD, SUB, MUL, DIV) y unarios (NEG) que pueden ser utilizados en el árbol de sintaxis abstracta.
Estas clases de constructores de tipos se utilizan para estructurar y definir los tipos de datos de manera clara y precisa en Haskell, lo que facilita la manipulación y el procesamiento de los datos en el código.