Tarea: Reducción entre Problemas NP-Completos

Teoría de la Computación

Junio 2025

Objetivo

El objetivo de esta tarea es verificar que un problema B pertenece a la clase NP-completo, mediante una reducción polinomial desde un problema clásico A ya conocido como NP-completo. En particular:

Problemas a estudiar

A: 3-SAT

Definido formalmente en la hoja de ejercicios

B: Optimización de Inversiones Internas con Dependencias entre Grupos de Proyectos

Una empresa tecnológica busca asignar de manera eficiente un presupuesto limitado para financiar iniciativas internas que permitan modernizar su infraestructura, reducir deuda técnica y optimizar procesos clave. Cada iniciativa está compuesta por proyectos, los cuales pueden pertenecer a uno o varios grupos temáticos (por ejemplo, "Aplicación de Clean Code", "Refactorización del Backend", "Migración a la Nube").

Cada proyecto tiene un costo y un valor estimado para la organización. La particularidad es que los proyectos pueden estar asociados a múltiples grupos, y la financiación de un solo proyecto implica que se debe financiar la totalidad de todos los grupos a los que pertenece ese proyecto. Es decir, si se decide financiar un proyecto, se debe financiar la colección completa de proyectos de cada grupo en el que dicho proyecto participa.

Por ejemplo, si un proyecto pertenece a los grupos "Automatización de QA" y "Estandarización de Infraestructura", entonces financiar ese proyecto implica financiar todos los proyectos de ambos grupos.

Por lo tanto, dado:

- Un conjunto de proyectos $P = \{p_1, p_2, \ldots, p_n\}$,
- Un conjunto de grupos $G = \{G_1, G_2, \ldots, G_m\}$, donde cada grupo $G_j \subseteq P$,
- Una función de costo $c : P \to N$,
- Una función de beneficio $b : P \to N$,
- Un presupuesto máximo $M \in N$,

- Un una cota inferior de beneficio $V \in N$,

con la restricción de que si se selecciona un proyecto p, entonces deben seleccionarse todos los proyectos de cada grupo al que p pertenece (financiación completa de grupos asociados),

¿Existe un subconjunto de proyectos $S \subseteq P$ tal que ...

1. La suma total de los costos de todos los proyectos seleccionados (incluidos aquellos añadidos por las dependencias grupales) no exceda M,
2. La suma total de los beneficios de los proyectos seleccionados sea al menos V,
3. Para cada proyecto $p \in S$, todos los proyectos en cada grupo al que p pertenece también estén en S.

Parte 1: Verificadores y Reducción en Haskell

1.1 Representación de dominios y soluciones

Definir en Haskell los siguientes tipos:

- Para A

```
type DomA = undefined
type SolA = undefined
```

- B:

```
type DomB = undefined
type SolB = undefined
```

1.2 Verificadores en tiempo polinomial

Implementar las siguientes funciones en Haskell:

```
verifyA :: (DomA, SolA) -> Bool
verifyB :: (DomB, SolB) -> Bool
```

Justificar formalmente que ambas funciones pueden evaluarse en tiempo polinomial respecto al tama no de entrada.

1.3 Resolución en tiempo exponencial

Implementar funciones que devuelvan una solución si existe, utilizando un algoritmo de exploración:

```
solveA :: DomA -> SolA
solveB :: DomB -> SolB
```

Estas funciones no deben ser eficientes; su objetivo es ilustrar que la solución puede encontrarse por fuerza bruta y ser verificada en tiempo polinomial.

1.4 Reducción polinomial entre problemas

Definir una estrategia para la reducción de instancias de A a instancias de B:

```
reduceAToB :: DomA -> DomB
```

No es necesario implementarla pero sí describirla formalmente. Debe justificarse que es efectivamente una reducción polinomial según la definición vista en el curso.

Parte 2: Verificadores y Reducción en otros modelos

2.1 Verificador de 3-SAT en Imp

Implementar en Imp puro el siguiente programa para la verificación de 3-SAT:

VERIFY_SAT(clausulas, valuacion)

que devuelve en una variable *res* el booleano correspondiente a la verificación de la instancia *clausulas* sobre la valuación *valuacion* (asumiendo en todas las variables estructuras apropiadas).

Puede asumir la función

lookup(valuacion, id) returns on valor

e invocarla según la sintaxis del Ejercicio 7 del Práctico 2.

2.2 Codificación de la Reducción en Máquinas de Turing

Definir un alfabeto Σ y una codificación de las instancias de `DomA` y `DomB` como cadenas sobre Σ.

Especificar (no programar) una Máquina de Turing que, al recibir como entrada una codificación de una instancia de `A`, produzca como salida la codificación de una instancia de `B` tal que:

- Si la instancia de `A` es satisfacible, entonces la instancia de `B` tiene una solución.
- Si la instancia de `A` no es satisfacible, entonces la instancia de `B` no tiene una solución.

Entrega

El contenido de la entrega consistirá en:

- Un archivo Haskell de nombre `Solucion.hs` para las secciones 1.1, 1.2, 1.3.
  Se valorará el uso de buenas prácticas en Haskell. Se deberá incluir tests que evidencien el cumplimiento de lo solicitado.
- Un documento PDF de nombre Solucion.pdf:
  - Encabezado con nombre completo y número de estudiante.
  - Parte 1: Para la Sección 1.4
  - Parte 2: Para las secciones 2.1, 2.2, 2.3

  Se valorará una buena redacción y justificación utilizando los conceptos y resultados vistos en el curso.

La fecha de entrega límite será el **2 de Julio a las 23:55** por Aulas.

Se permitirán entregas de **hasta dos estudiantes del mismo dictado**.