# CONTENIDO

# CONTENIDOS DE FIGURAS

# CONTENIDOS DE TABLAS

# Resumen

Este proyecto de tesis fue originado por la tesis doctoral de Jorge Alberto Ruiz Vanoye en la que se buscó proponer una transformación polinomial entre el problema de Bin-Packing a 2-Partition. Partiendo del conocimiento de que existía una transformación polinomial de 2-Partition a Bin-Packing, se pretendió realizar la transformación en sentido contrario, basándose en la definición de completitud-NP, según la cual debería de existir la transformación de Bin-Packing a 2-Partition. Sin embargo, los resultados no fueron del todo satisfactorios, ya que no se pudieron transformar correctamente todos los ejemplares de prueba.

Para comprender por qué fue imposible encontrar una transformación satisfactoria, se revisó la teoría de la completitud-NP y la definición de transformación polinomial. El resultado de dicha revisión fue el establecimiento de tres condiciones implícitas que debe cumplir cualquier transformación entre problemas NP-completos.

Al aplicar estas condiciones a la transformación 2-Partition a Bin-Packing, se encontró que no cumplía con una de las condiciones implícitas; por lo tanto, se decidió revisar bajo las mismas condiciones a la rama de transformaciones que van desde el problema de Bin-Packing hasta el problema de Satisfactibilidad (SAT) con el fin de verificar que cumplan dichas condiciones.

Todo parece indicar que no hay ningún error en las transformaciones hasta ahora revisadas, pero se ha encontrado que para algunas de ellas no ha sido posible encontrar una transformación revertida o transformación en sentido contrario. En la mayor parte de las demostraciones de completitud-NP de un problema sólo se incluye la transformación en un solo sentido, omitiendo la de sentido inverso, lo cual da a sospechar que tal vez dicha transformación no es posible o es muy difícil de diseñar.

Uno de los propósitos de la completitud-NP es establecer una relación entre los problemas NP-completos mediante la cual, si tal sólo se encuentra un algoritmo que resuelva polinomialmente cualquier problema NP-completo, entonces todos los problemas NP-completos también serán resueltos. El método para establecer esta relación entre dos problemas NP-completos es la transformación polinomial, y para cumplir con el propósito antes mencionado, es lógico pensar que debe poder transformarse en ambos sentidos.

Dado que el término de transformación inversa no está bien definido y que el término transformación está relacionada al de función, podría suponerse que el término de transformación inversa es idéntico al de función inversa. Por lo tanto, se propone el término de transformación revertida a una transformación polinomial entre dos problemas $\Pi_1$ y $\Pi_2$ NP-completos que va de $\Pi_2$ a $\Pi_1$ cuando ya existe una transformación polinomial de $\Pi_1$ a $\Pi_2$.

# Capítulo 1. Introducción

La complejidad computacional es la rama de la teoría de la computación que tiene como objetivo mostrar cuán complejo necesita ser un algoritmo en función del problema que trata de resolver [Garey & Johnson, 1979].

Dentro de la complejidad computacional se encuentra la teoría de la completitud-NP, la cual tiene como objetivo clasificar los problemas de decisión que no pueden ser resueltos por un algoritmo determinista en tiempo polinomial (clase NP). Exponiéndolo de otra manera, es el conjunto de problemas para los que no se ha encontrado un algoritmo polinomial que los resuelva pero que tampoco se ha comprobado que sean intratables (que no exista un algoritmo polinomial que los resuelva).

Un problema NP-completo tiene la propiedad de que puede ser resuelto en tiempo polinomial si y sólo si todos los problemas pertenecientes a la clase NP-completa pueden ser resueltos en tiempo polinomial. Suponiendo que un problema $A \in$ NP-completo fuera resuelto en tiempo polinomial, entonces cualquier problema $X \in$ NP-completo también sería resuelto en tiempo polinomial, con lo cual la teoría de la completitud-NP quedaría descartada.

## 1.1 Motivaciones

Dentro de las definiciones de la teoría de la completitud-NP se pueden inferir algunas propiedades que deben cumplir las transformaciones polinomiales para que un problema sea considerado NP-completo, y que tal vez no han sido tomadas en cuenta en la realización de transformaciones polinomiales que se encuentren en la literatura.

Una de estas propiedades indica que para cada par de problemas NP-completos $\Pi_1$ y $\Pi_2$, debe existir una transformación polinomial de $\Pi_1$ a $\Pi_2$, y de igual manera, debe existir una transformación revertida de $\Pi_2$ a $\Pi_1$. En la literatura hay muchos ejemplos de transformaciones polinomiales entre dos problemas NP-completos en un sentido, omitiendo la transformación en sentido contrario.

Es de interés encontrar una respuesta al por qué en muchas transformaciones no se ha reportado la de sentido contrario. De igual forma, es motivador poder dar una explicación al por qué, en algunos casos, los intentos de realizar transformaciones revertidas de una transformación polinomial conocida entre dos problemas NP-completos no han sido satisfactorios.

## 1.2 Justificación

La interpretación de la teoría de la completitud-NP en base a sus definiciones fundamentales hace suponer que la transformación entre dos problemas cualesquiera pertenecientes a esta clase es posible en ambos sentidos.

Uno de los principales fundamentos de los problemas NP-completos, es que todos los problemas pertenecientes a la clase NP-completa pueden transformarse polinomialmente a otro problema de la misma clase.

La importancia de las transformaciones polinomiales radica en los siguientes dos aspectos: Una transformación polinomial nos da la posibilidad de adaptar cualquier algoritmo que resuelva un problema $\Pi_1$ para que resuelva un problema $\Pi_2$ al igual que poder extrapolar los conocimientos de un problema $\Pi_1$ en un problema $\Pi_2$.

La teoría de la completitud-NP establece que si tan solo un problema NP-completo es resuelto por un algoritmo determinista en tiempo polinomial, entonces todos los problemas NP-completos también serán resueltos gracias a la transformación polinomial. Por lo anterior, es de suponerse que la transformación pueda realizarse en cualquier sentido entre cualquier par de problemas para cumplir con la condición antes mencionada.

Sin embargo, existen casos en los que no se ha podido realizar una transformación polinomial satisfactoria entre dos problemas pertenecientes a la clase NP-completa.

Éstos son indicios de que tal vez no haya una transformación válida entre dos problemas NP-completos cualesquiera. Es decir, no cualquier par de problemas NP-completos pueden ser transformados en ambos sentidos.

Por ello es importante la revisión de transformaciones entre problemas NP-completos existentes en la literatura, con el fin de determinar cuál es la causa de que este fenómeno se presente en ciertos pares de problemas.

## 1.3 Definición del problema

Existen indicios de una posible anomalía dentro de la teoría de la completitud-NP, específicamente se tiene la sospecha de la no pertenencia del problema de Partition a la clase NP-completa, por lo cual fue necesario realizar una revisión a las transformaciones polinomiales que involucren a este problema con el fin de verificar que cumplan las condiciones necesarias para ser válidas.

Parte de la dificultad del proyecto de tesis consiste en revisar el mayor número de transformaciones polinomiales entre pares de problemas conocidos pertenecientes a la

clase NP-completo, con el fin de verificar si la transformación es correcta y cumple con las características de un problema NP-completo.

## 1.4 Antecedentes

Al hablar de transformación polinomial y reducción es inevitable hacer alusión a Cook y Karp, primeros investigadores en realizar transformaciones polinomiales de ejemplares de un problema o lenguaje [Garey & Johnson, 1979 y R.Karp, 1972].

Entre los antecedentes se encuentra la tesis doctoral desarrollada por Jorge Alberto Ruiz Vanoye, en la cual se propuso una forma de redefinir de manera formal los ejemplares y transformaciones polinomiales de ejemplares de un problema NP-completo a otro problema NP-completo. Dichas transformaciones se enfocaron inicialmente a la transformación polinomial del problema de Bin-Packing al problema del agente viajero (PAV) con el fin de extrapolar indicadores de caracterización de Bin-Packing a algún otro problema NP-completo [Ruiz Vanoye, 2008].

Inicialmente, la tesis doctoral de Ruiz Vanoye planteaba transformaciones entre los problemas de Bin-Packing y agente viajero (PAV), pero se encontró que no existía en la literatura una transformación entre estos dos problemas en la literatura. Entonces, se propuso buscar una transformación indirecta, transformar mediante problemas intermedios. Se realizó una investigación en la literatura y se elaboró un grafo de transformaciones entre problemas reducibles a Bin-Packing y problemas reducibles a PAV pero al analizar el grafo se encontró que los problemas Bin-Packing y PAV se encontraban alejados entre sí y que en base a los sentidos de las transformaciones intermedias, parecía imposible llegar a realizar dicha transformación. En vista de esta dificultad, se decidió trabajar con la extrapolación de indicadores de Bin Packing a Partition.

Partiendo del conocimiento de que existía una transformación polinomial de 2-Partition a Bin-Packing, se pretendió realizar la transformación en sentido contrario, basándose en la definición de completitud-NP, de la cual se desprende la existencia de dicha transformación [Vanoye 2008, Ortega, Pazos].

En los experimentos realizados en dicha tesis, se transformaron 2214 ejemplares-sí de Bin-Packing en 1161 ejemplares-sí y 1053 ejemplares-no de 2-Partition, lo cual indica que el algoritmo de transformación tiene fallas.

El simple hecho de transformar ejemplares-sí en ejemplares-no invalida la transformación polinomial debido a que contradice la misma definición de transformación polinomial. Desafortunadamente, tras varios intentos, no se pudo realizar una corrección adecuada al algoritmo de transformación por lo que se planteó la siguiente interrogante, ¿es posible diseñar un algoritmo de transformación sin fallas que transforme del problema de Bin-Packing al problema de Partition en base en la teoría de la completitud-NP?

## 1.5 Objetivo general

Existen indicios de posibles anomalías en algunas transformaciones entre problemas NP-completos, lo cual podría repercutir en la pertenencia o no de algunos de estos problemas a la clase NP-completa.

El objetivo de esta tesis es poder determinar si existe en realidad esta anomalía dentro de las transformaciones polinomiales, específicamente las que se encuentran en la rama de transformaciones que van desde el problema de Partition al problema del SAT.

### 1.5.1 Objetivos particulares

- Revisar transformaciones polinomiales entre pares de problemas NP-completos.
- Detectar posibles anomalías en las transformaciones entre algunos problemas NP-completos conocidos y, si es posible, proponer correcciones.
- Verificar si el problema Partition pertenece a la clase NP-completa.

## 1.6 Alcances y limitaciones

Alcances:

- Establecer las condiciones necesarias que una transformación debe cumplir al transformar dos problemas NP-completos cualesquiera.
- Revisar las transformaciones polinomiales que conectan desde el problema Bin Packing hasta el problema de Satisfactibilidad (SAT).
- Determinar las anomalías, en caso de presentarlas, de cada una de las transformaciones revisadas.

Limitaciones:

- Se limita a revisar sólo una pequeña rama de transformaciones en las que se cree que existe una anomalía.
- Se limita la revisión a transformaciones encontradas en la literatura.

# Capítulo 2. Marco Teórico

En el desarrollo de técnicas para la revisión de diversos problemas NP-completos, es necesario establecer un marco teórico fundamental con el propósito de presentar las más importantes definiciones y los conceptos más fundamentales de la teoría de la completitud-NP que ayuden a la comprensión de este trabajo de tesis para aquellos lectores que no estén familiarizados con la teoría de la completitud-NP.

No se pretende dar una explicación profunda de tales definiciones, simplemente se presentan como apoyo al entendimiento del trabajo a tratar en esta tesis. Se recomienda acudir a las fuentes bibliográficas si se quiere indagar más en los conceptos que a continuación se presentan.

## 2.1 Problemas de decisión y optimización

La gran mayoría de los problemas de interés en aplicaciones en la vida real son problemas de optimización, en los cuales cada solución factible tiene un valor asociado, y en el que se desea encontrar una solución factible con el mejor valor posible [T.H. Cormen, 1990].

De forma más detallada, el ejemplar de un problema de optimización está compuesto por un conjunto de configuraciones (restricciones), una función objetivo que tiene como finalidad asignar un valor real a cada ejemplar. La respuesta es encontrar cuál de las soluciones factibles tiene el mejor valor de la función objetivo. Dependiendo de la definición del problema, el mejor valor puede ser el mayor valor objetivo encontrado (problema de maximización) o el menor valor objetivo encontrado (problema de minimización) [E. Schaeffer, 2007].

Un problema de decisión es aquél que sólo tiene dos posibles respuestas: sí o no, uno o cero [Garey & Jhonson, 1979].

En términos más formales se define un problema de decisión $\Pi(D, Y)$ como una pareja de un conjunto de ejemplares $D$ y un conjunto de ejemplares-sí $Y$ donde $Y \subseteq D$ [Garey & Johnson, 1979].

La teoría de la completitud-NP está diseñada para ser aplicada sólo a problemas de decisión; sin embargo, es importante observar que esta definición puede ser extendida a problemas de optimización, debido a que cada problema de optimización tiene asociado su problema de decisión. Usualmente se puede modelar un problema de decisión a partir de un problema de optimización imponiendo un límite al valor que será optimizado [T.H. Cormen, 1990].

Por ejemplo, considérese el problema de optimización SHORTEST-PATH, en donde dado un grafo no dirigido $G$ y un par de vértices $u$ y $v$, se desea encontrar un camino de $u$ a $v$ en el cual se visite el menor número posible de aristas. El problema de decisión asociado al problema SHORTEST-PATH es denominado PATH, el cual enuncia que dado un grafo dirigido $G$ y un par de vértices $u$ y $v$, y un entero $k$, ¿existe un camino de $u$ a $v$ que visite a lo más $k$ vértices? [T.H. Cormen, 1990].

En el problema de optimización se busca el menor valor objetivo, mientras que en el problema de decisión se busca una respuesta sí o no.

## 2.2 Codificación de un problema de decisión

Para que una máquina de Turing determinista pueda resolver un problema de decisión cualquiera, primero es necesario codificarlo para su procesamiento. Por lo anterior, es necesario definir el concepto de codificación de un problema.

Suponiendo que $\Sigma$ sea un conjunto de símbolos y que $\Sigma^*$ sea el conjunto de todas las cadenas posibles formadas por una secuencia de símbolos de $\Sigma$, entonces, la codificación de un problema de decisión $\Pi$ bajo un esquema de codificación $e$ se define de la siguiente manera [Garey & Johnson, 1979]:

$$L[\Pi, e] = \left\{ x \in \Sigma^* : \begin{array}{l} \Sigma \text{ es el alfabeto utilizado por } e, \text{ y } x \text{ es la} \\ \text{codificación bajo } e \text{ de un ejemplar } I \in Y \end{array} \right\}$$

## 2.3 Máquina de Turing Determinista (MTD) y la clase P

Una máquina de Turing determinista consta de los siguientes elementos:

a) Una cinta infinita bidireccional en la cual estarán escritos los símbolos de un ejemplar codificado de un problema de decisión.
b) Una cabeza de lectura-escritura que pueda leer los símbolos de la cinta infinita para poder pasarlos al control de estados finitos, que escriba los símbolos especificados por el control de estados finitos y que se mueva hacia la izquierda o a la derecha de la cinta.
c) Un control de estados finitos implementado como un autómata finito determinista que procese los símbolos leídos en la cinta [Garey & Johnson, 1979].

Un programa para una MTD (máquina de Turing determinista) cuenta con la siguiente información:

1. Un conjunto finito $\Sigma$ de símbolos de entrada y un conjunto $\Gamma = \Sigma \cup \{b\}$, donde $b$ denota un símbolo blanco.
2. Un conjunto finito $Q$ de estados, que incluye un estado inicial $q_0$ y tres estados de paro $q_Y$, $q_N$, y $q_I$ (que corresponden a los estados donde la MTD se detiene cuando la cadena de entrada codifica un ejemplar-sí, un ejemplar-no o un ejemplar inválido, respectivamente).
3. Una función de transición $\delta$: $(Q - [q_Y, q_N]) \times \Gamma \rightarrow Q \times \Gamma \times [-1, +1]$ [Garey & Johnson, 1979].

La función de transición recibe un estado del conjunto $Q$ diferente a los de paro ($q_Y$, $q_N$ y $q_I$) y un símbolo de entrada, y produce una triada que consiste de un estado (el nuevo estado de la MTD), un símbolo de salida escrito en la cinta y un movimiento de la cabeza (donde –1 y 1 indican el movimiento de la cabeza hacia la izquierda o hacia la derecha respectivamente) [Garey & Johnson, 1979].

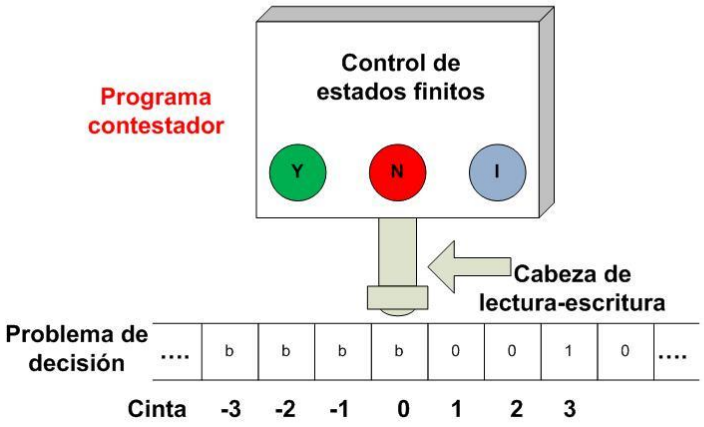



Figura 2.1.- Representación esquemática de una MTD. (Adaptado de [Garey & Johnson, 1979]).

## 2.4 Tiempo polinomial y tiempo exponencial

Para propósitos de la complejidad computacional, una diferencia polinomial en términos de tiempo de ejecución es considerada relativamente *pequeña*, al contrario de una diferencia exponencial considerada relativamente *grande* [M. Sipser, 2006].

Por ejemplo, considérese dos algoritmos con $A$ y $B$, tales que el algoritmo $A$ tiene un desempaño de $n^3$ mientras que el algoritmo $B$ tiene un desempaño de $2^n$. Para una entrada donde $n$ sea igual a 1000, el algoritmo $A$ tendría un desempeño de mil millones, un número considerablemente alto, pero manejable; sin embargo, en el algoritmo $B$ el

desempaño calculado es mucho más alto que el número total de átomos en el universo, una gran diferencia [M. Sipser, 2006].

En general, los algoritmos de tiempo exponencial generalmente se utilizan para resolver problemas realizando una búsqueda exhaustiva en el espacio de soluciones, a esta técnica se le llama *búsqueda de fuerza bruta* [M. Sipser, 2006].

Los modelos computacionales más aceptables generalmente son los polinomialmente equivalentes, aquéllos en los que se presenta un incremento polinomial en su tiempo de ejecución [M. Sipser, 2006]. En la actualidad la mayoría de los programadores se preocupan por desarrollar programas con tiempos de ejecución polinomiales cada vez más rápidos sin descuidar la eficiencia de los mismos.

La clase P está formada por todos los problemas de decisión que pueden ser resueltos por una máquina de Turing determinista en tiempo polinomial. En términos más formales, la definición de la clase P es [Garey & Johnson, 1979]:

P = {$L$: existe una MTD de tiempo polinomial para un programa $M$ tal que $L = L_M$}

Un problema de decisión $\Pi$ pertenece a P bajo un esquema de codificación *e* si $L$ [$\Pi$,*e*] $\in$ P; es decir, si existe una MTD de tiempo polinomial que resuelva $\Pi$ bajo el esquema de codificación *e* [Garey & Johnson, 1979].

La clase P es de gran importancia dentro de la teoría de la complejidad debido a:

1. P es invariante para todos los modelos computacionales que son polinomialmente equivalentes a una máquina de Turing determinista y
2. P corresponde aproximadamente a la clase de problemas que realmente se pueden resolver en una computadora.

## 2.5 Máquina de Turing no determinista (MTND) y la clase NP

El modelo de una máquina de Turing no determinista (MTND) es casi el mismo que el de una MTD, excepto por la incorporación de un módulo de "adivinación" que tiene su propia cabeza de escritura.

La MTND cuenta con los siguientes elementos:

a) Una cinta infinita bidireccional, igual a la de la MTD.
b) Una cabeza escritora, que escriba los símbolos provenientes del módulo de "adivinación" sobre la cinta bidireccional.
c) Un modulo de adivinación, el cual es implementado como un autómata finito no determinista. Este módulo genera aleatoriamente una solución candidata para el problema de decisión. También controla la cabeza escritora que imprime en la cinta los símbolos de la solución candidata.

d) Una cabeza de lectura-escritura, igual a la de la MTD.
e) Un control de estados finitos que verifica si una solución candidata es en efecto una solución para el problema de decisión.

Entonces, de manera informal, la clase NP está conformada por todos los problemas de decisión que pueden ser resueltos por una MTND de tiempo polinomial. Definiendo formalmente la clase NP tenemos:

NP = {$L$: existe una MTND de tiempo polinomial para un *programa* $M$ *tal* que $L = L_M$} [Garey & Johnson, 1979].

Un problema de decisión $\Pi$ se dice que pertenece a la clase NP bajo un esquema de codificación *e* si el lenguaje $L$ [$\Pi$, $e$] $\in$ P; es decir, si existe una MTND de tiempo polinomial que resuelva $\Pi$ bajo el esquema de codificación *e* [Garey & Johnson, 1979].

Otra forma de ver a la clase NP es como el conjunto de lenguajes que pueden ser *verificados* en tiempo polinomial.

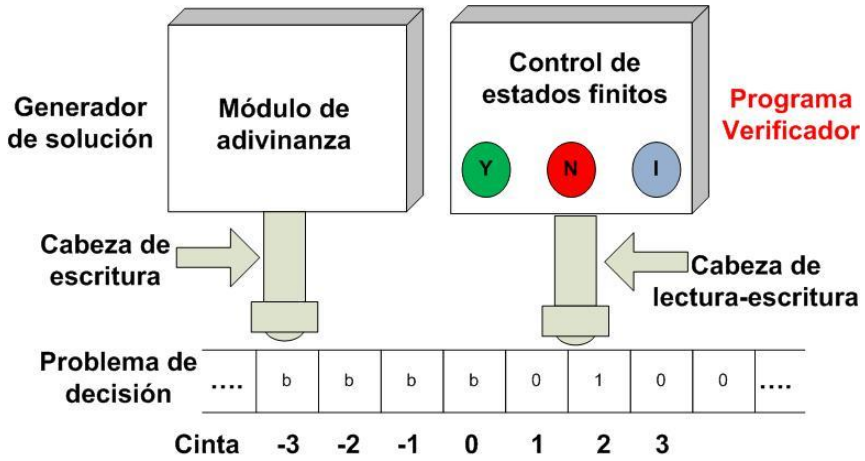



Figura 2.2.- Representación esquemática de una MTND. (Adaptado de [Garey & Johnson, 1979]).

## 2.6 La relación entre P y NP

Para empezar, P $\subseteq$ NP, por lo que cualquier problema $\Pi$ que pueda ser resuelto en tiempo polinomial por una MTD también puede ser resuelto en tiempo polinomial por una MTND. "***La comprobación de lo anterior puede verse cuando usamos un algoritmo determinista en la etapa de verificación para un algoritmo no determinista***". Si existe un problema $\Pi$ $\in$ P y una MTD $A$ de tiempo polinomial para $\Pi$, entonces se puede obtener una MTND de tiempo polinomial para $\Pi$ al aplicar $A$ en la etapa de verificación. Entonces se tiene que $\Pi$ $\subseteq$ P implica que $\Pi \subseteq$ NP, lo cual se ilustra en la figura 2.3.

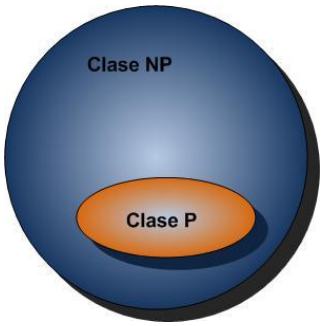



Figura 2.3.- Representación gráfica de la relación entre la clase P y la clase NP (Adaptado de [Garey & Johnson, 1979]).

## 2.7 Teoría de la completitud-NP

En 1971, Stephen Cook estableció las bases de lo que hoy se conoce como la teoría de la completitud-NP. Dado un problema de decisión $\Pi$, si P $\neq$ NP, entonces $\Pi =$ NP $-$ P por lo que $\Pi \in$ NP-completo.

La teoría de la completitud-NP provee varias técnicas para demostrar que un problema dado "es tan difícil" de resolver como una larga lista de otros problemas conocidos. Un problema se definirá como una pregunta general a ser contestada, el cual generalmente posee varios parámetros, o variables libres cuyos valores aún no están especificados. Un problema está descrito por:

a) una descripción general de todos los parámetros y
b) una declaración de qué propiedades son necesarias satisfacer.

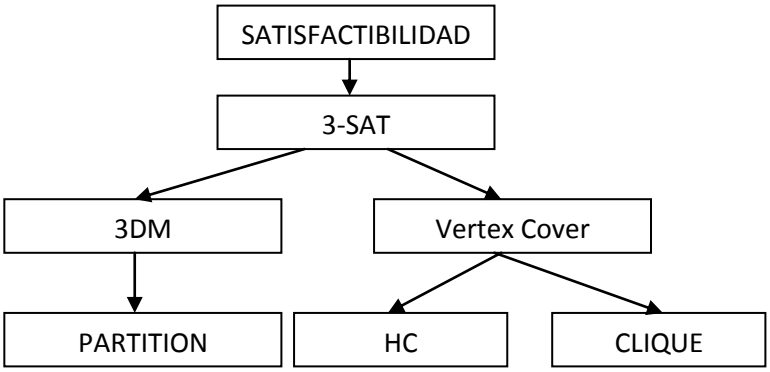



Figura 2.4.- Diagrama de secuencia de transformaciones para comprobación de completitud-NP (Adaptado de [Garey & Johnson, 1979]).

Un ejemplar de un problema es una asignación de valores a todos los parámetros de un problema.

La base fundamental de la teoría de la completitud-NP es el problema de Satisfactibilidad, dado que es la base para la comprobación de que un problema dado es NP-completo. Esto se realiza al reducir un problema dado al problema de SAT. De esta forma, dado un problema $\Pi$, si $\Pi \propto$ SAT entonces $\Pi \in$ NP-completo.

Entonces, la teoría de la completitud-NP parte de la creencia de que P $\neq$ NP, por lo cual, si se demuestra que por lo menos un problema perteneciente a NP-completo puede ser resuelto por un algoritmo determinista de tiempo polinomial, entonces P = NP = NP – C y el teorema de la completitud-NP quedaría obsoleto.

De manera formal, un lenguaje $L$ pertenece a la clase NP-completa si $L \in$ NP y, para todos los otros lenguajes $L' \in$ NP, $L' \propto L$ [Garey & Johnson, 1979].

Cabe señalar que la teoría de la completitud-NP no demuestra que un problema $\mathbf{\Pi}$ es intratable, sino que sólo demuestra que un problema $\Pi_1$ es al menos tan difícil como un problema $\Pi_2$.

## 2.8 Demostración de la completitud-NP

Para probar que un problema $\Pi_1$ pertenece a NP-completo se siguen los siguientes pasos:

1. Demostrar que $\Pi_1 \in$ NP.
2. Demostrar que $\Pi_1 \in$ NP-completo [Garey & Johnson, 1979].

Para demostrar de que $\Pi_1 \in$ NP, debe existir una MTND de tiempo polinomial que resuelva $\Pi_1$.

Para demostrar que $\Pi_1 \in$ NP-completo, primero es necesario encontrar un problema $\Pi_2$, tal que $\Pi_2 \in$ NP-completo. Después se formula una función de reducción de $\Pi_1 \propto \Pi_2$. Por último demostrar que la función de reducción $\Pi_1 \propto \Pi_2$ es polinomial [Garey & Johnson, 1979].

## 2.9 Definiciones de reducción polinomial

**Definición 2.9.1.-** Garey and Johnson definen de manera informal reducción polinomial como “una transformación constructiva que mapea cualquier ejemplar de un primer problema en un ejemplar equivalente de un segundo problema. Esta transformación proporciona los medios para convertir cualquier algoritmo que resuelva un segundo problema en un algoritmo correspondiente para resolver un primer problema”.

**Definición 2.9.2.-** La definición de reducción polinomial fue dada por primera vez por Stephen Cook, quien establece que "un conjunto $S$ de caracteres es **reducible-P** (P de polinomial) a un conjunto $T$ de caracteres si y sólo si alguna máquina de consulta $M$ y un polinomio $Q(n)$ tal que para cada cadena de entrada $w$, la computación-$T$ de $M$ con entrada $w$ se detiene en $Q(|w|)$ pasos (donde $|w|$ es la longitud de $w$), y termina en un estado de aceptación si y sólo si $w \in S$" [S. A. Cook, 1971].

**Definición 2.9.3.-** Karp define: "Sean $L$ y $M$ un par de lenguajes, decimos que $L \in M$ ($L$ es reducible a $M$) si existe una función $f \in \Pi$ tal que $f(x) \in M \longleftrightarrow x \in L$" [R. Karp, 1972].

Con base en lo anterior, Karp introduce la siguiente definición de completitud: "Si $M \in$ P y $L \propto M$ entonces $L \in$ P. Decimos entonces que $L$ y $M$ son equivalentes si $L \propto M$ y $M \propto L$. Decimos que $L$ es completo si $L \in$ NP y cada lenguaje de NP es reducible a $L$" [R. Karp, 1972].

Un lenguaje $L_1$ es reducible a un lenguaje $L_2$ ($L_1 \leq_p L_2$) si existe una función $f$ computable en tiempo polinomial, para toda $x \in L_1$ si y sólo si $f(x) \in L_2$ [D. M. Mount, 2003].

## 2.10 Transformación polinomial

**Definición 2.10.1.-** Una transformación polinomal de un lenguaje $L_1 \subseteq \Sigma_1^*$ a un lenguaje $L_2 \subseteq \Sigma_2^*$ es una función $f : \Sigma_1^* \longrightarrow \Sigma_2^*$ que satisface las siguientes dos condiciones:

1. Existe una MTD de tiempo polinomial que computa $f$.
2. Para toda $x \in \Sigma_1^*$, $x \in L_1$ si y sólo si $f(x) \in L_2$ [Garey & Johnson, 1979].

(Notas: La transformación polinomial que va de $L_1$ a $L_2$ se denota usualmente por $L_1 \propto L_2$. A las transformaciones se les llama usualmente reducciones. Para ver ejemplos de transformaciones ir al capítulo 5.)

Considerando la transformación polinomial $f$ de $L_1$ a $L_2$, entonces la transformación revertida de $f$ se define como una función $\phi : \Sigma_2^* \rightarrow \Sigma_1^*$ que satisface las siguientes dos condiciones:

1. Existe una MTD de tiempo polinomial que computa $\phi$.
2. Para toda $y \in \Sigma_2^*$, $y \in L_2$ si y sólo si $\phi(y) \in L_1$.

Nótese que si $f$ es una transformación polinomial que va de $L_1$ a $L_2$ y $\phi$ es la transformación revertida de $f$, para cada cadena dada $x \in \Sigma_1^*$, en general no necesariamente ocurre que $\phi(f(x)) = x$. Sin embargo, dada una transformación polinomial $f$

de $L_1$ a $L_2$, definimos el término *transformación inversa* $f^{-1}$ como una transformación revertida tal que $f^{-1}(f(x)) = x$.

Una transformación polinomial entre un problema $\Pi_1$, que se conoce es NP-completo y un problema $\Pi_2$ implica que ejemplares-sí de un problema $\Pi_1$ puedan transformarse a ejemplares-sí de un problema $\Pi_2$.

Un ejemplar de un problema es un conjunto de valores asignados a cada uno de los parámetros del problema. Si la respuesta al problema es sí, entonces es un ejemplar-sí. Por el contrario, si la respuesta al problema es no, entonces es un ejemplar-no.

De lo anterior podemos concluir que un problema de decisión está compuesto por ejemplares-sí y ejemplares-no. Además, debemos considerar a los ejemplares inválidos, que son aquellos ejemplares que no satisfacen los requisitos de la definición del problema.

# Capítulo 3. Estado del Arte

En las conclusiones de la tesis doctoral de Jorge Alberto Ruiz Vanoye, se menciona la detección de posibles anomalías en las definiciones establecidas sobre transformación polinomial, específicamente sobre posibles diferencias entre el libro de Garey & Johnson de 1979 y los artículos de Karp y Cook en lo que refiere a la teoría de transformación polinomial.

También sugiere como trabajos a futuro realizar un análisis sobre la teoría de la completitud-NP expuesta en el libro de Garey & Johnson y en los artículos de los autores originales.

Lo anterior mencionado es el motivo principal de este trabajo: proponer, en caso de ser necesario, una corrección a la teoría de la completitud-NP.

Actualmente, la teoría de la transformación polinomial está basada en el libro de Garey & Johnson publicado en 1979, el cual toma los conceptos de los artículos de Karp y Cook, los autores originales del concepto de transformación polinomial.

## 3.1 Stephen A. Cook(1971)

En 1971 Stephen A. Cook publicó su artículo, en el que se fundamentan los conceptos de lo que hoy conocemos como la teoría de la completitud-NP, al igual que la idea de transformación polinomial, en lo que él llamaba *P-reducibility*.

En su definición (**Definición 2.9.2** del marco teórico), Cook maneja dos conjuntos de caracteres $S$ y $T$, una máquina de consulta $M$ y un polinomio $Q(n)$, una cadena de caracteres $w$ que es computada por $M$ y se detiene en $Q(|w|)$ pasos y termina en un estado de aceptación si $w \in S$.

En el artículo de Cook no hace diferencia entre ejemplares-sí y ejemplares-no, es más, no utiliza el término ejemplar.

## 3.2 Richard M. Karp (1972)

Richard M. Karp toma los conceptos de Cook para definir su función de reducción, en donde introduce el término de lenguaje.

También define la clase NP como el conjunto de todos los lenguajes reconocibles en tiempo polinomial por una máquina de Turing no determinista. De igual manera utiliza el término $\Pi$ como la clase de todas las funciones de $\Sigma^*$ a $\Sigma^*$ computables en tiempo polinomial por una máquina de Turing.

Karp establece que un lenguaje es reducible a otro si existe una función $f \in \Pi$ tal que $f(x) \in M$ si y sólo si $x \in L$.

En su artículo aún no se manejan los términos de ejemplar ni hace distinción entre los ejemplares-sí y los ejemplares-no. Maneja una cadena de caracteres pertenecientes a un alfabeto.

## 3.3 Garey & Johnson (1979)

En el libro de Garey & Johnson es donde por primera vez se usa el término de transformación polinomial, que es definido como una función polinomial $f: \Sigma_1^* \rightarrow \Sigma_2^*$, la cual involucra una máquina de Turing determinista que pueda computarla tal que para toda $x \in \Sigma_1^*$, $x \in L_1$ si y sólo si $f(x) \in L_2$.

Define $\Sigma$ como el alfabeto (conjuntos de símbolos), $\Sigma^*$ como todas las cadenas de símbolos de $\Sigma$, $L$ es el lenguaje y es un subconjunto de $\Sigma^*$.

Garey & Jonhson consideran que un lenguaje $L$ es *NP-completo*, si $L$ es *NP*, y para cualquier otro lenguaje $L'$ que sea *NP-completo,* $L' \propto L$.

Dentro de las definiciones de transformación polinomial, se menciona que si $L_1 \propto L_2$ y $L_2 \propto L_1$, entonces $L_1$ y $L_2$ son polinomialmente equivalentes.

## 3.4 Sipser (2006)

Otra definición de transformación polinomial es expresada en [Sipser, 2006], la cual define:

Una función $f: \Sigma^* \rightarrow \Sigma^*$ es una *función computable en tiempo polinomial* si existe alguna máquina de Turing de tiempo polinomial $M$ que pare en justo $f(w)$ pasos, empezando con una entrada $w$.

Un lenguaje $A$ es polinomialmente reducible a un lenguaje $B$, escrito $A \leq_p B$, si existe una función computable en tiempo polinomial $f: \Sigma^* \rightarrow \Sigma^*$, donde para cada $w$,

$$w \in A \leftrightarrow f(w) \in B$$

## 3.5 Trabajos relacionados al tema

Hasta el momento no se ha encontrado trabajo alguno en el que propongan correcciones a las definiciones de la teoría de la completitud-NP, o que se mencione sobre indicios de posibles anomalías en estas definiciones, ni tampoco existe evidencia en la literatura que pudiera ofrecer una explicación de dichas anomalías. Sin embargo, existen trabajos que exponen posibles debilidades en diferentes ramas de la complejidad computacional sin formalizar nada aún y sólo exponiendo como posibles debilidades o limitaciones de la teoría de la complejidad.

Arnold L. Rosenberg menciona en su trabajo [A.L. Rosenberg, 2010] la existencia de limitaciones dentro de la teoría de la complejidad computacional, a lo cual llama debilidades. Tras replantearse la manera de abordar los fundamentos matemáticos de la complejidad computacional, menciona que la teoría acepta como computables muchas funciones de las que no se tiene idea cómo computar (por ejemplo, la función "run-of-7s") y acepta como computables otras funciones que, aunque computables en un universo conceptual con recursos infinitos, no son computables en un universo real con recursos limitados. Aunque no existe relación entre este trabajo y los temas tratados en esta tesis, es un indicativo de algunas limitantes en la teoría de la complejidad.

Otra observación en la complejidad computacional es la realizada en [R. Greenlaw, 1995], donde se trata la teoría de la completitud-P (la rama de la teoría de la complejidad que se centra en identificar los problemas más duros en la clase P). Un problema P-completo es aquél que está en NP y todo problema en P puede reducirse a él en tiempo polilogarítmico en una máquina paralela con un número polinomial de procesadores. Las soluciones altamente paralelas para los problemas P-completos son deficientes, esto es, para problemas de la clase P para los que es fácil encontrar soluciones secuenciales, es difícil encontrarles soluciones factibles altamente paralelas en tiempo polilogarítmico usando un numero polinomial de procesadores. Lo anterior contradice al supuesto de que entre más procesadores para dar solución a un problema, más rápido se obtiene la solución. Cabe mencionar que la completitud-P es muy distinta a la completitud-NP; por lo tanto, el trabajo anteriormente mencionado es muy diferente de los temas tratados en esta tesis.

Luca Trevisan, en su tesis doctoral [L. Trevisan, 1997] estudió las reducciones de conservación de aproximación entre problemas de optimización combinatoria para probar los resultados de completitud en clases de aproximación. Durante su trabajo utilizó las diferentes reducciones de conservación de aproximabilidad existentes hasta el momento y demostró que ninguna de ellas permitía comprobar ciertos resultados de completitud naturales. Demostró que la reducción-L, la reducción más usada en la literatura, tenía muy malas propiedades de completitud tal como la de no preservar el factor constante de

aproximabilidad entre problemas. Aunque la tesis mencionada trata sobre ciertas clases de problemas (concretamente, clases de aproximabilidad); sin embargo, el tema tratado en esa tesis es diferente del tema de la clase de problemas NP-completos tratado en esta tesis.

Los trabajos mencionados anteriormente son una muestra que dentro de la teoría de la complejidad computacional existen limitantes tanto teóricas como practicas a las que no se les ha podido encontrar solución, por lo cual nos hace pensar que es poco probable que la teoría de la completitud-NP esté exenta de estas limitaciones.

Hasta el momento, y muy probablemente, el único trabajo del que se tiene conocimiento en el que se mencionan indicios de estas posibles anomalías es la tesis doctoral de Jorge Alberto Ruiz Vanoye, en la cual se abordó el tema de transformación de ejemplares e indicadores entre los problemas NP-completos Bin-Packing y Partition.

Por lo anteriormente mencionado, esta tesis está proponiendo un tema que no ha sido tratado por otros investigadores; y por lo tanto, no se tienen referencias adicionales sobre trabajos relacionados.

# Capítulo 4. Condiciones para validar una transformación entre problemas NP-completos

## 4.1 Introducción

Uno de los avances importantes dentro del campo de la complejidad computacional y en específico sobre la relación entre la clase P y la clase NP, se dio en los inicios de los años 70s con el trabajo de Stephen Cook y Leonid Levin. Ellos descubrieron ciertos problemas en la clase NP cuya complejidad individual está relacionada con la de toda la clase [Sipser, 2006].

Por el momento, en la teoría más aceptada se supone que P $\neq$ NP, en donde P $\subseteq$ NP; y por lo tanto, los problemas más difíciles en NP no se encuentran en P. Estos problemas son llamados NP-completos [A. Maheshwari, 2010].

Una de las propiedades fundamentales de los problemas NP-completos, ya sea en lo teórico o en lo práctico, es que si se encuentra un algoritmo de tiempo polinomial que resuelva cualquier problema NP-completo, entonces también todos los problemas de esta clase estarán resueltos en tiempo polinomial con lo que la teoría de P $\neq$ NP quedaría anulada y daría paso a P = NP [M. Sipser, 2006]. Esta propiedad es dada gracias a la relación que tiene los problemas NP-completos.

En la teoría de la completitud-NP, para comprobar que un problema $\Pi_1$ pertenece a la clase NP-completa, es necesario que cumpla con las siguientes condiciones:

1. Que el problema $\Pi_1$ sea NP.
2. Para cualquier otro problema $\Pi_2$ que pertenezca a NP, $\Pi_2$ es reducible a $\Pi_1$ [Garey & Jhonson, 1979].

En el primer paso, para comprobar que un problema es NP, es necesario demostrar que existe una máquina de Turing no determinista que "resuelva" el problema en tiempo polinomial. Esto puede ser expresado mediante un algoritmo o programa que genere soluciones para el problema y verifique si es una solución aceptada.

En el segundo paso se hace uso de la reducción, la cual es una técnica de la complejidad computacional mediante la cual podemos convertir un problema a otro problema de tal manera que la solución utilizada para el primer problema nos sirva también para el segundo problema [M. Sipser, 2006].

La reducción es comúnmente usada para clasificar los problemas en complejidad computacional. Por ejemplo, supongamos que tenemos un problema $A$ y un problema $B$,

sabemos que el problema $A$ pertenece a un tipo de problemas $T$, por lo tanto, si $B$ es reducible a $A$, entonces $B$ también es un problema del tipo $T$. La misma función tiene la reducción en la teoría de la completitud-NP.

## 4.2 Condiciones explícitas de una transformación

La transformación utilizada para comprobar que un problema $\Pi_1$ pertenece a la clase de problemas NP-completos es la transformación polinomial. Podemos definir a una transformación polinomial como una función que transforma un problema a otro en tiempo polinomial.

Formalmente podemos definir una transformación polinomial de la siguiente manera [A. Maheshwari, 2010]:

*"Sean $A \subseteq \{0,1\}^*$ y $B \subseteq \{0,1\}^*$ dos lenguajes. Diremos que $A \leq_p B$ si existe una función*

$$f: \{0,1\}^* \rightarrow \{0,1\}^*$$

*tal que:*

1. $f \in FP$
2. *para todas las cadenas w en {0,1}*,*

$$w \in A \leftrightarrow f(w) \in B"$$

En donde $\leq_p$ simboliza la reducción polinomial y FP significa que sea una función que compute en tiempo polinomial.

Otra definición es dada por [M. Sipser, 2006] en la que menciona lo siguiente:

*"Un lenguaje $A$ es polinomialmente reducible a un lenguaje $B$, escrito $A \leq_p B$, si existe una función computable en tiempo polinomial $f: \Sigma^* \rightarrow \Sigma^*$, donde para cada w,*

$$w \in A \leftrightarrow f(w) \in B"$$

Podemos observar que ambas definiciones son muy similares. Para el propósito de este trabajo de investigación se tomará como base la definición de transformación polinomial en [Garey and Jhonnson, 1979].

> *"Una transformación polinomal de un lenguaje $L_1 \subseteq \Sigma_1^*$ a un lenguaje $L_2 \subseteq \Sigma_2^*$ es una función $f : \Sigma_1^* \rightarrow \Sigma_2^*$ que satisface las siguientes dos condiciones"*:
>
> A. *Existe una MTD de tiempo polinomial que computa $f$ .*
> B. *Para toda $x \in \Sigma_1^*$, $x \in L_1$ si y sólo si $f(x) \in L_2$.*

Tomando la segunda condición (B) de la transformación polinomial, ésta se puede reescribir de la siguiente manera:

B. Para toda $x \in \Sigma_1^*$,
   1. si $f(x) \in L_2$ entonces $x \in L_1$ (**si $x \notin L_1$ entonces $f(x) \notin L_2$**) y
   2. **si $x \in L_1$ entonces $f(x) \in L_2$** (si $f(x) \notin L_2$ entonces $x \notin L_1$).

Finalmente, para el propósito de estudio de este proyecto, simplificamos la condición como se muestra a continuación:

B. Para toda $x \in \Sigma_1^*$,
   1. **si $x \in L_1$ entonces $f(x) \in L_2$ y**
   2. **si $x \notin L_1$ entonces $f(x) \notin L_2$.**

Y ocasionalmente, se usarán las siguientes expresiones alternativas:

B. Para toda $x \in \Sigma_1^*$,
   **1'. si $f(x) \in L_2$ entonces $x \in L_1$ y**
   **2'. si $f(x) \notin L_2$ entonces $x \notin L_1$.**

## 4.3 Condiciones implícitas de una transformación

Durante el proceso de revisión de las definiciones fundamentales de la teoría de la completitud-NP, se dedujeron algunas características que deben cumplir las transformaciones polinomiales y que no se encuentran explícitas en las definiciones.

Las transformaciones polinomiales entre dos problemas NP-completos $\Pi_1$ y $\Pi_2$ deben cumplir con tres condiciones implícitas, además de las condiciones explícitas que se encuentran en las definiciones de la teoría de la completitud-NP.

Las tres condiciones implícitas son:

C1. Todos los ejemplares del problema $\Pi_1$ deben poder ser transformados en ejemplares del problema $\Pi_2$. Más específicamente, todos los ejemplares-sí del problema $\Pi_1$ deben ser transformados en ejemplares-sí del problema $\Pi_2$, y todos los ejemplares-no e inválidos de $\Pi_1$ deben ser transformados en ejemplares-no e inválidos de $\Pi_2$.

C2. Cada ejemplar en el problema $\Pi_1$ debe transformarse en un ejemplar equivalente en el problema $\Pi_2$.

C3. La transformación debe poder realizarse en ambos sentidos, es decir, de $\Pi_1$ a $\Pi_2$ y de $\Pi_2$ a $\Pi_1$.

### 4.3.1 Primera condición implícita: transformación de todos los ejemplares

La primera condición implícita se deriva de la definición de transformación polinomial (de lenguajes) [Garey and Johnson, 1979].

Donde $L_1$ y $L_2$ son las codificaciones de dos problemas de decisión $\Pi_1$ y $\Pi_2$ respectivamente. En base a lo anterior se puede observar que la condición B establece que debe existir una transformación para todas y cada una de las cadenas del lenguaje $L_1$, es decir, el problema de decisión $\Pi_1$. Para comprobar que esto es así, supóngase que se definiera una transformación $L_1 \propto L_2$ tal que para alguna $x' \in L_1$ su imagen $f(x')$ no existiera, entonces en este caso no se cumpliría la condición B, y por lo tanto, dicha transformación no sería válida.

Este hecho es reconocido por Garey&Johnson (p. 13) en el siguiente comentario: "La principal técnica usada para demostrar que dos problemas están relacionados es la de reducir uno a otro, dando un transformación constructiva que mapee cualquier ejemplar del primer problema a un ejemplar equivalente del segundo". En la última oración, la frase cualquier ejemplar implica todos los ejemplares.

Conviene tener presente que la codificación de un problema $\Pi$ en un lenguaje bajo el esquema de codificación $e$ se define de la siguiente manera [Garey&Johnson, 1979]:

$$L[\Pi, e] = \left\{ x \in \Sigma^* : \begin{array}{l} \Sigma \text{ es el alfabeto usado por } e, \text{ y } x \text{ es la codi-} \\ \text{ficación bajo } e \text{ de un ejemplar } I \in Y_\Pi \end{array} \right\}$$

lo cual implica la división del conjunto $\Sigma^*$ de todas las cadenas en dos subconjuntos: el conjunto de cadenas-sí y el conjunto de cadenas-no, de tal manera que el conjunto de cadenas-sí corresponde al conjunto de ejemplares-sí de $\Pi$, mientras que el conjunto de cadenas-no se divide en dos subconjuntos: uno que corresponde al conjunto de ejemplares-no de $\Pi$, y otro que corresponde al conjunto de ejemplares inválidos de $\Pi$.

Por lo tanto, en base a la revisión de las definiciones anteriormente dadas, se puede concluir que para que una transformación de un problema $\Pi_1$ a un problema $\Pi_2$ sea válida, todos los ejemplares-sí de $\Pi_1$ deben ser transformados en ejemplares-si de $\Pi_2$ y todos los ejemplares-no e inválidos de $\Pi_1$ deben ser transformados en ejemplares-no e inválidos de $\Pi_2$.

### 4.3.2 Segunda condición implícita: transformación en ejemplares equivalentes

La segunda condición implícita proviene de la esencia de la transformación polinomial, la cual se expresa de la siguiente manera [Garey & Johnson, 1979]:

> La principal técnica usada para demostrar que dos problemas están relacionados es la de *reducir* uno a otro, dando una transformación constructiva que mapee cualquier ejemplar del primer problema a un *ejemplar equivalente* del segundo. Dicha transformación proporciona el medio para convertir cualquier algoritmo que resuelva el segundo problema en un algoritmo correspondiente para resolver el primer problema.

La forma directa de convertir un algoritmo $A_2$ que resuelve un segundo problema a otro algoritmo $A_1$ para resolver un primer problema, consiste en la implementación de dos fases: En la primera fase se convierte un ejemplar $p_1$ del primer problema a un ejemplar $p_2$ del segundo problema, y en la segunda fase se utiliza el algoritmo $A_2$ para determinar si el ejemplar $p_2$ es un ejemplar-sí; finalmente, si la transformación de $p_1$ a $p_2$ es una transformación polinomial, entonces, por la condición B1', también se puede determinar si el ejemplar $p_2$ es un ejemplar-sí o no.

### 4.3.3 Tercera condición implícita: transformación en ambos sentidos

La definición de la tercera condición implícita se deduce de la definición de la completitud-NP [Garey & Johnson, 1979].

> Formalmente, un lenguaje *L* se define como NP-completo si $L \in$ NP y, para todos los otros lenguajes $L' \in$ NP, $L' \propto L$.

Para tal efecto, considérense dos problemas NP-completos $\Pi_1$ y $\Pi_2$ cuyas codificaciones son $L_1$ y $L_2$ respectivamente; entonces, haciendo primero $L$=$L_1$ y $L'$=$L_2$, la definición establece que debe cumplirse que $L_2 \propto L_1$; después, haciendo $L$=$L_2$ y $L'$=$L_1$, la definición establece que debe cumplirse que $L_1 \propto L_2$. En conclusión, estos dos resultados implican que los problemas deben poder transformarse en ambos sentidos.

# Capítulo 5. Revisión de transformaciones entre problemas NP-completos

## 5.1 Introducción

El propósito de este trabajo de tesis, como ya se ha mencionado anteriormente, es poder encontrar la explicación al fenómeno que se presenta en varias de las transformaciones polinomiales que se conocen en la teoría de la completitud-NP. Como es sabido, una de las propiedades principales de la transformación entre problemas NP-completos es que debe poder realizare en ambos sentidos. Para cualquier par de problemas NP-Completos $A$ y $B$, si existe una transformación de $A$ a $B$ también debe existir una transformación de $B$ a $A$. Sin embargo, en la literatura la gran mayoría de las demostraciones de pertenencia de problemas a la clase NP-completa sólo incluyen la transformación en un sentido, dando por entendido que es posible realizarla en sentido inverso. Este fenómeno en particular fue detectado en la transformación entre los problemas Partition y Bin Packing en la que no se pudo encontrar una transformación en sentido contrario que cumpliera con todas las condiciones necesarias para considerarse válida.

Con el fin de poder encontrar la causa de este fenómeno se decidió revisar la rama de transformaciones que van desde el problema Partition al problema SAT, el cual es el primer problema demostrado que es NP-completo.

Una posible hipótesis para explicar esta anomalía es que el problema de Partition pudiera no ser NP-completo. Para tal efecto, se decidió revisar la rama de transformaciones que van desde el problema de Partition al problema SAT, el cual es el primer problema demostrado que es NP-completo. Específicamente, la verificación de la completitud-NP de Partition, implica la revisión de tres transformaciones: SAT $\propto$ 3-SAT, 3-SAT $\propto$ 3DM, y 3DM $\propto$ Partition. La revisión tiene por objeto detectar algún posible error en algunas de las transformaciones, en particular el hecho de que alguna no satisfaga alguna de las condiciones implícitas (C1, C2, C3). El hecho de que alguna transformación fuera incorrecta no es inconcebible, ya que algunas transformaciones (por ejemplo en [R.Karp, 1972]), estas no son verificadas rigurosamente. A este respecto conviene mencionar que una transformación de Partition a Knapsack (descrita en la sección 5.6) es incorrecta, ya que no satisface una de las condiciones implícitas.

## 5.2 La revisión de la transformación entre los problemas de Partition y Bin Packing

La posible anomalía encontrada en la transformación entre los problemas Partition y Bin Packing consiste en lo siguiente: es posible encontrar una transformación polinomial de Partition a Bin-Packing pero parece imposible encontrar una de Bin Packing a Partition, con lo cual se estaría incumpliendo con la condición C3 (Capítulo 4: Condiciones implícitas).

Para tratar de explicar esta anomalía, primero se revisará la transformación polinomial existente en la literatura de Partition a Bin Packing, y luego se describirán diversos intentos de transformación polinomial de Bin Packing a Partition con el fin de mostrar la dificultad de encontrar una transformación en sentido contrario.

Antes de adentrarnos en la revisión de la transformación entre estos dos problemas, primero daremos un vistazo a sus definiciones.

**Problema Partición:** Considérese un conjunto finito $A$ y un tamaño $s(a) \in Z^+$ para cada elemento $a \in A$ (donde $Z^+$ denota los enteros positivos), entonces el problema Partition consiste en determinar si existe un subconjunto $A' \subset A$ tal que $\Sigma_{a \in A'} s(a) = \Sigma_{a \in A - A'} s(a)$.

**Problema Bin-Packing:** Considérese un conjunto $U$ de objetos, un tamaño $s(u) \in Z^+$ para cada $u \in U$ (donde $Z^+$ denota los enteros positivos), una capacidad de contenedor $B$ entera positiva y un entero positivo $K$, entonces el problema Bin-Packing consiste en determinar si existe una partición de $U$ en subconjuntos disjuntos $U_1$, $U_2$, ..., $U_K$ tal que la suma de los tamaños de los objetos en cada $U_i$ sea igual a $B$ o menor.

### 5.2.1 Algoritmo Trans Par-BP:

0. Entrada: un ejemplar (sí, no o inválido) $p$ del problema Partition.

1. Para cada elemento $a_i$ en Partition definir un objeto $u_i$ en Bin-Packing tal que $s(u_i) = s(a_i)$. (Notas: Con esto quedan definidos todos los objetos de Bin-Packing, incluyendo sus tamaños en términos de los parámetros de Partition. La complejidad de este paso es $O(|A|)$ y, por lo tanto, polinomial.)

2. Hacer $S = \Sigma_{a \in A} s(a)$ y hacer $B = S/2$. (Notas: Con esto queda definida la capacidad de los contenedores de Bin-Packing. La complejidad de este paso es $O(|A|)$ y, consecuentemente, polinomial.)

3. Hacer $K = 2$. (Notas: Con esto queda definido el número de contenedores de Bin-Packing. La complejidad de este paso es $O(1)$ y, por tanto, polinomial.)

### 5.2.2 Verificación de las condiciones explícitas

Para verificar la validez de esta transformación, primero se considerarán las condiciones explícitas A y B de la definición de transformación polinomial (Capítulo 4: Condiciones explícitas).

Primeramente, como cada uno de los pasos del algoritmo tiene complejidad polinomial, entonces el algoritmo completo satisface la condición A.

En segundo lugar, para la condición B hay que tomar en cuenta que $L_1$ es el conjunto de las cadenas que codifican a los ejemplares-sí de Partition mientras que $L_2$ es el de las cadenas que codifican a los ejemplares-sí de Bin-Packing.

Para verificar la condición B1 considérese un ejemplar cualquiera $p$ de Partition, entonces si $p$ es un ejemplar-sí la suma de los tamaños de todos sus elementos ($S = \Sigma_{a \in A}\, s(a)$) debe ser par y, consecuentemente, se puede aplicar el paso 2 del algoritmo para obtener la capacidad $B$ de los contenedores para un ejemplar $q$ de Bin-Packing que es la imagen de $p$; además, con los elementos de Partition debe ser posible formar dos grupos tal que la suma de los tamaños de los elementos en cada grupo sea $S/2$ (= $B$). Finalmente, esto implica que en el ejemplar $q$ debe ser posible formar dos subconjuntos tales que cada subconjunto contenga los objetos correspondientes a los elementos de un grupo distinto del ejemplar $p$ y que la suma de los tamaños de sus elementos sea $B$, lo cual a su vez implica que $q$ sea un ejemplar-sí de Bin-Packing, con lo cual se verifica que se cumple la condición B1.

Para la verificación de la condición B2, se utilizará la prueba por contradicción; por lo tanto, supóngase que si $x \notin L_1$ entonces $f(x) \in L_2$. Si $x \notin L_1$, entonces esto implica que $x$ es una cadena-no de $\sum_1{}^*$, y con lo cual nos lleva a dos posibles casos: que $x$ sea una codificación de un ejemplar inválido de Partition o que $x$ sea una codificación de un ejemplar-no de Partition. A continuación se describen estos dos casos.

- Primer caso: Supóngase que $q$ fuera un ejemplar-sí de Bin-Packing y que fuera la imagen de un ejemplar inválido $p$ de Partition. El problema Partition no tiene muchas opciones para generar ejemplares inválidos; así que el que $p$ fuera inválido implicaría que el tamaño de por lo menos uno de sus elementos $a_i$ no fuera un entero positivo; entonces, al aplicar el paso 1 del algoritmo de transformación a $p$, resultaría que la imagen $q$ de $p$ tendría un objeto $u_i$ cuyo tamaño tampoco sería entero positivo, lo cual implicaría que $q$ fuera inválido, y consecuentemente, el supuesto de este caso es imposible.
- Segundo caso: Supóngase que $q$ fuera un ejemplar-sí de Bin-Packing y que fuera la imagen de un ejemplar-no $p$ de Par. Nótese que existen dos subconjuntos de ejemplares-no de Par: aquéllos en los que $S = \Sigma_{a \in A}\, s(a)$ es impar y aquéllos en los que $S$ es par, lo cual da origen a dos subcasos.
  - Supóngase ahora que $S$ fuera impar; esto implicaría que al aplicar el algoritmo de transformación a $p$, resultaría que en la imagen $q$ de $p$ el tamaño de los

contenedores sería un número fraccionario, lo que implicaría que $q$ fuera inválido, y consecuentemente, el supuesto de este caso es imposible.

- Supóngase que $S$ fuera par, entonces como $p$ es un ejemplar-no, sucede que para cualquier posible partición de los elementos de Partition (en $A'$ y $A–A'$) debe suceder que $\Sigma_{a\in A'}\ s(a) > S/2$ o $\Sigma_{a\in A–A'}\ s(a) > S/2$, esto implicaría que al aplicar el algoritmo de transformación a $p$, resultaría que en la imagen $q$ de $p$ para cada posible partición de los objetos de Bin-Packing (en $U_1$ y $U_2$) también debe suceder que $\Sigma_{u\in U1}\ s(u) > B$ o $\Sigma_{u\in U2}\ s(u) > B$, lo que implicaría que *q* fuera un ejemplar-no, y consecuentemente, el supuesto de este subcaso es imposible.

Ahora que ha quedado demostrado que la transformación Trans Par-BP cumple con las condiciones explícitas, es necesario verificar también que cumpla con las condiciones explícitas.

### 5.2.3 Verificación de las condiciones implícitas

Con relación a la condición implícita C1, obsérvese que el paso 0 del algoritmo de transformación toma como entrada cualquier ejemplar (sí, no o inválido) del problema Partition, y los pasos 1, 2 y 3 del algoritmo permiten transformar cualquier ejemplar de Partition, con lo cual se comprueba que la transformación satisface la condición C1.

Respecto a la condición implícita C2, supóngase que tuviéramos un ejemplar $p$ de Partition y un algoritmo $A_{BP}$ que resuelva cualquier ejemplar de Bin-Packing (en donde "resolver" un ejemplar quiere decir responder "si" o "no" para un ejemplar); entonces, es posible diseñar un algoritmo $A_{Par}$ basado en el algoritmo $A_{BP}$ que pueda resolver $p$. El algoritmo $A_{Par}$ constará de dos fases: la primera fase usará el algoritmo Trans Par-BP para transformar el ejemplar $p$ en un ejemplar $q$ de Bin Packing, y la segunda fase usa el algoritmo $A_{BP}$ para resolver el ejemplar $q$. Como se ha comprobado que la transformación cumple la condición explícita B, si la respuesta del algoritmo $A_{BP}$ es "*sí*" para el ejemplar $q$, entonces, por la condición B1' (si $f(x) \in L_2$ entonces $x \in L_1$) el ejemplar $p$ debe ser también un ejemplar-sí, de lo contrario, $p$ debe ser un ejemplar-no o inválido. La transformación cumple con la condición C2.

En cuanto a la condición implícita C3, veremos que parece imposible satisfacerla con cualquier transformación revertida (por ejemplo de Bin-Packing a Partition) lo cual constituye una paradoja, ya que se debería poder satisfacerla según la teoría de la completitud-NP dado que Partition y Bin-Packing son problemas NP-completos.

Primeramente, el diseño de una transformación de la transformación inversa de Trans Par-BP es una de las tareas que nos permitiría determinar si en realidad existe este problema. Analizando el algoritmo Trans Par-BP se puede observar que el paso 1 se puede

invertir; sin embargo, los pasos 2 y 3 no, por lo tanto, el algoritmo quedaría de la siguiente manera.

### 5.2.4 Algoritmo Trans BP-Par1:

0. Entrada: un ejemplar (sí, no o inválido) $q$ del problema Bin-Packing.

1. Para cada objeto $u_i$ en Bin-Packing definir un elemento $a_i$ en Partition tal que $s(a_i) = s(u_i)$. (Notas: Con esto quedan definidos todos los elementos de Partition, incluyendo sus tamaños en términos de los parámetros de Bin-Packing. La complejidad de este paso es $O(|U|)$ y, por lo tanto, polinomial.)

Para poder demostrar que la condición B1 no se cumple, basta con encontrar un caso que viole la condición. Considere un ejemplar-sí $q$ de Bin-Packing en el que $K = 5$, $B = 3$, y un conjunto de objetos $U = \{u_1, u_2, u_3, u_4, u_5, u_6, u_7, u_8, u_9, u_{10}\}$ con sus respectivos tamaños $s(u_1)=2$, $s(u_2)=1$, $s(u_3)=2$, $s(u_4)=1$, $s(u_5)=2$, $s(u_6)=1$, $s(u_7)=2$, $s(u_8)=1$, $s(u_9)=2$, $s(u_{10})=1$. Nótese que la partición $U_1=\{u_1, u_2\}$, $U_2=\{u_3, u_4\}$, $U_3=\{u_5, u_6\}$, $U_4=\{u_7, u_8\}$, $U_5=\{u_9, u_{10}\}$ constituye una solución de $q$, y por consiguiente es un ejemplar-sí. Al aplicar el algoritmo de transformación Trans BP-Par1 al ejemplar $q$, se obtiene el ejemplar $p$, el cual consta de un conjunto de elementos $A = \{a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8, a_9, a_{10}\}$ con sus respectivos tamaños $s(a_1)=2$, $s(a_2)=1$, $s(a_3)=2$, $s(a_4)=1$, $s(a_5)=2$, $s(a_6)=1$, $s(a_7)=2$, $s(a_8)=1$, $s(a_9)=2$, $s(a_{10})=1$. Puede verse que no es posible dividir $A$ en dos subconjuntos $A'$ y $A–A'$, tales que $\Sigma_{a\in A'} s(a) = \Sigma_{a\in A-A'} s(a)$, ya que $\Sigma_{a\in A} s(a)$ es impar, lo cual significa que $p$ es un ejemplar-no, y por consiguiente, se viola la condición B1.

Este problema que presenta la transformación Trans BP-Par1 puede ser corregido mediante una pequeña modificación al algoritmo de transformación. El algoritmo quedará reescrito de la siguiente manera.

### 5.2.5 Algoritmo Trans BP-Par2:

0. Entrada: un ejemplar (sí, no o inválido) $q$ del problema Bin-Packing.

1. Para cada objeto $u_i$ en Bin-Packing definir un elemento $a_i$ en Partition tal que $s(a_i) = s(u_i)$. (Notas: Con esto quedan definidos todos los elementos de Partition, incluyendo sus tamaños en términos de los parámetros de Bin-Packing. La complejidad de este paso es $O(|U|)$ y, por lo tanto, polinomial.)

2. Si $\Sigma_{u \in U}\, s(u)$ es impar, agregar al conjunto $A$ de Partition un elemento extra $\alpha$ con tamaño $s(\alpha)$=1. (Nota: La complejidad de este paso es $O(|U|)$ y, por lo tanto, polinomial.)

A pesar de las correcciones realizadas al algoritmo Trans BP-Par1, aún sigue violando la condición B1: desafortunadamente lograr que la suma de los tamaños de los objetos sea par no garantiza que un ejemplar de Partition tenga solución como lo muestra el siguiente caso. Considérese el ejemplar-sí de Bin-Packing tal que $K$=5, $B$=11, y un conjunto de objetos $U$ = {$u_1$, $u_2$, $u_3$, $u_4$, $u_5$, $u_6$, $u_7$, $u_8$, $u_9$, $u_{10}$} con sus respectivos tamaños $s(u_1)$=10, $s(u_2)$=1, $s(u_3)$=10, $s(u_4)$=1, $s(u_5)$=10, $s(u_6)$=1, $s(u_7)$=10, $s(u_8)$=1, $s(u_9)$=10, $s(u_{10})$=1. Nótese que la partición $U_1$={$u_1$, $u_2$}, $U_2$={$u_3$, $u_4$}, $U_3$={$u_5$, $u_6$}, $U_4$={$u_7$, $u_8$}, $U_5$={$u_9$, $u_{10}$} constituye una solución de $q$, y por lo tanto es un ejemplar-sí. Al aplicar el algoritmo de transformación Trans BP-Par2 al ejemplar $q$, se obtiene el ejemplar $p$, el cual consta de un conjunto de elementos $A$ = {$a_1$, $a_2$, $a_3$, $a_4$, $a_5$, $a_6$, $a_7$, $a_8$, $a_9$, $a_{10}$, $a_{11}$} con sus respectivos tamaños $s(a_1)$=10, $s(a_2)$=1, $s(a_3)$=10, $s(a_4)$=1, $s(a_5)$=10, $s(a_6)$=1, $s(a_7)$=10, $s(a_8)$=1, $s(a_9)$=10, $s(a_{10})$=1, $s(a_{11})$=1. Puede verse que no es posible dividir $A$ en dos subconjuntos $A'$ y $A$–$A'$, tales que $\Sigma_{a \in A'}\, s(a) = \Sigma_{a \in A-A'}\, s(a)$, lo cual significa que $p$ es un ejemplar-no, y por consiguiente, se viola la condición B1.

Para superar este problema se propone el siguiente algoritmo que garantiza la obtención de un ejemplar-sí de Partition.

### 5.2.6 Algoritmo Trans BP-Par3:

0. Entrada: un ejemplar (sí, no o inválido) $q$ del problema Bin-Packing.

1. Para cada objeto $u_i$ en Bin-Packin definir un elemento $a_i$ en Partition tal que $s(a_i) = s(u_i)$.

2. En caso de que no sea posible encontrar dos subconjuntos $A'$ y $A$–$A'$, tales que $\Sigma_{a \in A'}\, s(a) = \Sigma_{a \in A-A'}\, s(a)$, agregar al conjunto $A$ de Partition un elemento extra $\alpha$ con un tamaño $s(\alpha) \in Z^+$ tal que $\Sigma_{a \in A'}\, s(a) = \Sigma_{a \in A-A'}\, s(a) + s(\alpha)$ para alguna partición de $A$ en $A'$ y $A$-$A'$.

Aunque el algoritmo Trans BP-Par3 garantiza la satisfacción de la condición explícita B1, desafortunadamente ahora viola la condición A. Con respecto al paso 2 de Trans BP-Par3, determinar que no es posible encontrar los dos subconjuntos $A'$ y $A$–$A'$ tal que $\Sigma_{a \in A'}\, s(a) = \Sigma_{a \in A-A'}\, s(a)$ es equivalente a determinar si el ejemplar de Partition es un ejemplar-no, lo cual en general es tan complejo como determinar si un ejemplar de Partition es un

ejemplar-sí; por lo tanto, la complejidad del paso 2 no es polinomial, y por consiguiente, se viola la condición explícita A.

Para corregir este problema, se propone el siguiente algoritmo que garantiza la obtención de un ejemplar-sí de Partition.

### 5.2.7 Algoritmo Trans BP-Par4:

0. Entrada: un ejemplar (sí, no o inválido) $q$ del problema Bin-Packing.

1. Para cada objeto $u_i$ en Bin-Packing definir un elemento $a_i$ en Partition tal que $s(a_i) = s(u_i)$.

2. Agregar al conjunto $A$ de Partition un elemento extra $\alpha$ con un tamaño $s(\alpha) = \Sigma_{u \in U}\, s(u)$. (Nota: La complejidad de este paso es $O(|U|)$ y, por lo tanto, polinomial.)

3. Si $B \notin Z^+$ o $K \notin Z^+$ hacer $s(\alpha) = -1$. (Notas: La complejidad de este paso es $O(2)$. Es necesario agregar este paso para asegurar que los ejemplares inválidos de Bin-Packing se transformen en ejemplares inválidos de Partition.)

Aunque este algoritmo garantiza el cumplimiento de la condición A y B1, viola la condición explícita B2. Comprobaremos la violación de esta condición mediante la prueba por contradicción; entonces, supondremos que $x \notin L_1$ si y sólo si $f(x) \in L_2$. Para este caso, si $x \notin L_1$ quiere decir que $x$ es una cadena-no de $\Sigma_1$*, lo cual nos deja dos posibles casos: que $x$ sea la codificación de un ejemplar inválido de Bin-Packing o que $x$ sea la codificación de un ejemplar-no de Bin-Packing. Para el primer caso la condición B2 se cumple pero para el segundo caso no.

Para verificar el segundo caso, supongamos que $p$ fuera un ejemplar-sí de Partition y que fuera la imagen de un ejemplar-no $q$ de Bin-Packing. En este caso se mostrará que sí existe un caso que se ajusta a esta suposición, lo cual implicaría que no se satisface la condición B2. Para tal efecto considérese un ejemplar-no $q$ de Bin-Packing definido de la siguiente manera: $K = 3$ y $B = 11$, y $U = \{u_1, u_2, u_3, u_4, u_5, u_6\}$ con sus respectivos tamaños $s(u_1)=11$, $s(u_2)=1$, $s(u_3)=10$, $s(u_4)=1$, $s(u_5)=10$, $s(u_6)=1$. Al aplicar el algoritmo Trans BP-Par4 a $q$ se obtiene el ejemplar $p$ definido de la siguiente manera: $A = \{a_1, a_2, a_3, a_4, a_5, a_6, \alpha\}$ con sus respectivos tamaños $s(a_1)=11$, $s(a_2)=1$, $s(a_3)=10$, $s(a_4)=1$, $s(a_5)=10$, $s(a_6)=1$, $s(\alpha)=34$, con lo cual puede verse claramente que $p$ es un ejemplar-sí de Partition, lo cual implica que no satisface la condición explícita B2.

La violación de la condición B2 es originada por la aplicación del paso 2 del algoritmo Trans BP-Par4, ya que siempre agrega un elemento $\alpha$ con tamaño igual a la suma de los tamaños de los objetos del ejemplar de Bin-Packing; es decir, agrega $\alpha$ tanto para ejemplares-sí como para ejemplares-no. Desafortunadamente, no es fácil corregir este problema, ya que si se quisiera aplicar el paso 2 solamente a los ejemplares-sí, entonces sería necesario resolver primero el ejemplar de Bin-Packing, para lo cual no se conoce ningún algoritmo que pueda hacerlo en tiempo polinomial.

## 5.3 La revisión de la transformación entre los problemas de 3DM y Partition

La siguiente transformación dentro de la rama de transformaciones que van desde el problema del Bin-Packing hasta SAT es la de 3DM a Partition. Primero revisaremos la definición del problema de 3DM.

**3-Dimensional Matching (3DM):** Dados tres conjuntos $W = \{w_1, w_2, ..., w_q\}$, $X = \{x_1, x_2, ..., x_q\}$, $Y = \{y_1, y_2, ..., y_q\}$ donde $|W| = |X| = |Y| = q$, y una relación ternaria $M \subseteq X \times Y \times W$ de $k$ tripletas, el problema de 3DM consiste en determinar si existe un subconjunto $M' \subseteq M$ de $q$ tripletas de tal manera que cada elemento $\{w_i, x_i$ e $y_i\}$ aparezca sólo una vez en las tripletas de $M'$.

El siguiente algoritmo es una descripción resumida del proceso de transformación de 3DM a Partition [Garey & Johnson, 1979].

### 5.3.1 Algoritmo Trans 3DM-Par:

0. Entrada: un ejemplar (sí, no e inválido) $q$ del problema 3DM.

1. Si el ejemplar es inválido entonces genera un ejemplar inválido de Partition y parar el proceso, en caso contrario continuar con el paso 2.

2. Hacer el valor de $p$ igual a $\lceil \log_2(k+1) \rceil$.

3. Para cada tripleta $m_i = (w_{f(i)}, x_{g(i)}, y_{h(i)}) \in M$, definir un elemento $a_i \in A$ de Partition, cuyo tamaño estará dado por la siguiente fórmula:

$$s(a_i) = 2^{p(3q-f(i))} + 2^{p(2q-g(i))} + 2^{p(q-h(i))}$$

   donde $f(i)$, $g(i)$ y $h(i)$ son los subíndices de cada componente de la tripleta $m_i$.

4. finalmente, se definen dos elementos extras $b_1$ y $b_2$ de Partition, cuyos tamaños están dados por las siguientes expresiones:

$$s(b_1) = 2\sum_{i=1}^{k} s(a_i) - \sum_{j=0}^{3q-1} 2^{pj} \qquad s(b_2) = \sum_{i=1}^{k} s(a_i) + \sum_{j=0}^{3q-1} 2^{pj}$$

Es importante mencionar que $\sum_{i=0}^{3q-1} 2^{pj} = \sum_{a \in A'} s(a) = B$ si y sólo si $M'$ $\{m_i: a_i \in A'\}$ es un matching de 3DM.

(Nota: debido a la complejidad de este algoritmo, se recomienda al lector acudir a la fuente [Garey & Johnson, 1979] para mayor comprensión del mismo.)

### 5.3.2 Verificación de las condiciones explícitas

Para verificar la condición explícita A basta con observar el pseudoalgoritmo anterior para darse cuenta que todos los pasos tienen una complejidad polinomial, por lo cual se cumple la primera condición.

Para la condición B, hay que tomar en cuenta que $L_1$ es el conjunto de todas las cadenas que codifican a los ejemplares-sí de 3DM mientras que $L_2$ es el conjunto de todas las cadenas que codifican a los ejemplares-sí de Partition.

En la verificación de la condición B1, considérese un ejemplar-sí $b$ de 3DM, entonces si $b$ es un ejemplar-sí debe existir un conjunto $M'$ que no tenga elementos repetidos en las $q$ tripletas que lo componen; y por lo tanto, al aplicar el algoritmo 3DM-Par para obtener $d$ tenemos que $B = \sum_{i=0}^{3q-1} 2^{pj} = \sum_{a \in A'} s(a)$. Se crean los elementos de balanceo $s(b_1) = 2\sum_{i=1}^{k} s(a_i) - B$ y $s(b_2) = \sum_{i=1}^{k} s(a_i) + B$ los cuales se suman a los subconjuntos $A'$ y $A–A'$. Debido a que $\sum_{a \in A'} s(a) = B$ ambos subconjuntos ($A'$ y $A–A'$) sumarán exactamente el doble del tamaño del conjunto $A$ ($2\sum_{a \in A} s(a)$). Primero se realiza la suma del primer elemento de balanceo ($s(b_1)$) con el subconjunto $A'$ ($2\sum_{i=1}^{k} s(a_i) - B + \sum_{a \in A'} s(a)$) en donde al eliminar términos iguales ($B$ y $\sum_{a \in A'} s(a)$) nos queda un valor de $2\sum_{i=1}^{k} s(a_i)$ que es igual a $2\sum_{a \in A} s(a)$. Después se realiza la suma del segundo elemento de balanceo ($s(b_2)$) con el subconjunto $A–A'$ dando como resultado la siguiente operación: $\sum_{i=1}^{k} s(a_i) + B + \sum_{a \in A-A'} s(a)$ donde $B$ es el complemento de $\sum_{a \in A-A'} s(a)$ y la suma de ambos es igual a $\sum_{i=1}^{k} s(a_i)$ y por lo tanto, obtenemos un valor de $2\sum_{i=1}^{k} s(a_i)$ que es igual a $2\sum_{a \in A} s(a)$. Ambos elementos se balancean gracias a que $B = \sum_{a \in A'} s(a)$, y por lo tanto, sí es posible encontrar un emparejamiento en $M$ para poder crear el subconjunto $M'$ haciendo imposible que *b* sea la imagen de un ejemplar-no *d* de Partition, con lo cual se verifica que se cumple la condición B1.

Para la verificación de la condición B2, se utilizará la prueba por contradicción; por lo tanto, supóngase que si $x \notin L_1$ entonces $f(x) \in L_2$. Si $x \notin L_1$, entonces esto implica que $x$ es una cadena-no de $\sum_1{}^*$, y con lo cual nos lleva a dos posibles casos: que $x$ sea una codificación de un ejemplar inválido de 3DM o que $x$ sea una codificación de un ejemplar-no de 3DM.

- Primer caso: para el caso de que $x$ sea la codificación de un ejemplar inválido, el algoritmo de transformación genera un ejemplar inválido de Patition.
- Segundo caso: supongamos que tenemos un ejemplar-sí $d$ de Partition que fuera una imagen de un ejemplar-no $b$ de 3DM. Un ejemplar-no de 3DM es aquél en el cual falta al menos uno de los $q$ elementos de cualquiera de los conjuntos $W$, $X$, $Y$ en las tripletas del conjunto $M$. Entonces no sería posible encontrar $q$ tripletas sin elementos repetidos que conformen el conjunto $M'$. Por lo tanto, la suma de los pesos de las tripletas de $M'$ no coincidiría con el número $B$, y por lo tanto, a pesar de los dos elementos de balanceo, no se igualarían los pesos de $A'$ y $A–A'$ y por ende el ejemplar $d$ sería un ejemplar-no, lo cual contradice la suposición si $x \notin L_1$ entonces $f(x) \in L_2$. Se comprueba que la transformación cumple con la condición B2.

### 5.3.3 Verificación de las condiciones implícitas

Con respecto a la primera condición C1, el pseudoalgoritmo toma como entrada cualquier ejemplar de 3DM (sea ejemplar-sí, ejemplar-no o ejemplar inválido). El algoritmo transforma los ejemplares-sí de 3DM a ejemplares-sí de Partition. También transforma ejemplares-no de 3DM en ejemplares-no de Partition.

Para verificar la condición C2 primero supongamos que tenemos un ejemplar $b$ de 3DM y un algoritmo $A_{\text{Par}}$ que puede resolver cualquier ejemplar de Partition (donde resolver un ejemplar significa responder "sí" o "no" para este ejemplar). Entonces, supongamos que obtenemos un ejemplar $d$ de Partition al aplicar a $b$ el algoritmo de transformación, y después utilizamos el algoritmo $A_{\text{Par}}$ para resolver el ejemplar $d$. Como se ha comprobado que la transformación cumple la condición explícita B, si la respuesta del algoritmo $A_{\text{Par}}$ es "*sí*" para el ejemplar $d$, entonces, por la condición B1' (si $f(x) \in L_2$ entonces $x \in L_1$) el ejemplar $b$ debe ser también un ejemplar-sí, de lo contrario, $b$ debe ser un ejemplar-no o inválido. Por lo tanto, la transformación cumple con la condición C2.

Con respecto a la condición implícita C3, de la misma forma que en la revisión de la transformación entre Partition y Bin-Packing, no se ha podido demostrar la existencia de una transformación polinomial de Partition a 3DM; de igual forma, de acuerdo con nuestras revisión bibliográfica, no se ha reportado dicha transformación en la literatura sobre la teoría de la completitud-NP.

## 5.4 La revisión de la transformación entre los problemas de 3-SAT y 3DM

La comprobación de la pertenecía del problema de 3-Dimensional Matching a la clase NP-completa se realizó mediante la reducción de 3-SAT a 3DM. La definición del problema del 3-SAT se presenta a continuación.

**3-SATISFACTIBILIDAD (3-SAT):** Dada una colección $C = \{c_1, c_2, \ldots, c_m\}$ de cláusulas booleanas, donde cada cláusula está compuesta por exactamente tres variables de un conjunto $U = \{u_1, u_2, \ldots, u_n\}$, entonces, el problema consiste en determinar si existe una asignación de valores ($V$ o $F$) para cada $u_i$ ($i$=1, 2, ... , $n$), tal que $c_j$ sea verdadera para toda $j$ ($j$=1, 2, ... , $m$).

El siguiente algoritmo es una descripción resumida del proceso de transformación de 3-SAT a 3DM [Garey & Johnson, 1979].

### 5.4.1 Algoritmo Trans 3-SAT-3DM:

0. Entrada: un ejemplar (sí, no o inválido) $p$ del problema 3-SAT.

1. Si el ejemplar es inválido ($\exists\, c_j \in C$: $|c_j| \neq 3$) entonces generar un ejemplar inválido $q$ de 3DM y parar el proceso, en caso contrario, continuar con el paso 2.

2. Se define el siguiente conjunto de parámetros del problema de 3DM: $\{u_{i,j}, \neg u_{i,j}: 1 \leq i \leq n, 1 \leq j \leq m\}$, donde $n$ y $m$ representan el número de variables booleanas y el número de cláusulas booleanas del problema 3-SAT. Los parámetros de este conjunto formarán los elementos del conjunto $W$ del problema de 3DM.

3. Se define el siguiente conjunto de parámetros del problema de 3DM: $A = \{a_{i,j}: 1 \leq i \leq n, 1 \leq j \leq m\}$, $S_1 = \{s_{1,j}: 1 \leq j \leq m\}$, $G_1 = \{g_{1,j}: 1 \leq k \leq m(n-1)\}$. Los parámetros de estos conjuntos formarán los elementos del conjunto $X$ del problema de 3DM; es decir,
$$X = A \cup S_1 \cup G_1$$

4. Se define el siguiente conjunto de parámetros del problema de 3DM: $B = \{b_{i,j}: 1 \leq i \leq n, 1 \leq j \leq m\}$, $S_2 = \{s_{2,j}: 1 \leq j \leq m\}$, $G_2 = \{g_{2,j}: 1 \leq k \leq m(n-1)\}$. Los parámetros de estos conjuntos formarán los elementos del conjunto $Y$ del problema de 3DM; es decir,
$$Y = B \cup S_2 \cup G_2$$

5. Se construye el conjunto de tripletas *truth-setting and fan-out* (el cual es un subconjunto del conjunto $M$ de 3DM) de la siguiente manera:
$$T = \left( \bigcup_{i=1}^{n} T_i^t \right) \cup \left( \bigcup_{i=1}^{n} T_i^f \right)$$
donde:
$$T_i^t = \{(\neg u_{i,j}, a_{i,j}, b_{i,j}): 1 \leq j \leq m\}$$
$$T_i^f = \{(u_{i,j}, a_{i,j+1}, b_{i,j}): 1 \leq j \leq m\}$$
donde los elementos de las tripletas son tomados de los conjuntos $W$, $A$ y $B$ definidos en los pasos 1, 2 y 3.

6. Se construye el conjunto de tripletas *satisfaction testing* (el cual es un subconjunto del conjunto $M$ de 3DM) de la siguiente manera:

$$C = \bigcup_{j=1}^{m} C_j$$

donde:

$$C_j = \{(u_{i,j}, s_{1,j}, s_{2,j}) : u_i \in c_j\} \bigcup \{(u_{i,j}, s_{1,j}, s_{2,j}) : u_i \in \neg c_j\}$$

donde los elementos de las tripletas son tomados de los conjuntos $W$, $S_1$ y $S_2$ definidos en los pasos 1, 2 y 3.

7. Se construye el conjunto de tripletas *garbage collection* (el cual es un subconjunto del conjunto $M$ de 3DM) de la siguiente manera:

$$G = \{(u_{i,j}, g_{1,k}, g_{2,k}), (\neg u_{i,j}, g_{1,k}, g_{2,k}): 1 \le k \le m(n-1), 1 \le i \le n, 1 \le j \le m\}$$

donde los elementos de las tripletas son tomados de los conjuntos $W$, $G_1$ y $G_2$ definidos en los pasos 1, 2 y 3.

8. Se construye un subconjunto $M$ del problema de 3DM de la siguiente manera:

$$M = T \bigcup C \bigcup G$$

(Nota: debido a la complejidad de este algoritmo, se recomienda al lector acudir a la fuente [Garey & Johnson, 1979] para mayor comprensión del mismo.)

### 5.4.2 Verificación de las condiciones explícitas

Para verificar la condición explícita $A$ basta con observar el pseudoalgoritmo anterior para darse cuenta que todos los pasos tienen una complejidad polinomial. En la construcción de los subconjuntos $W$, $X$, $Y$, cada uno se lleva en $2mn$ pasos, por lo cual se cumple la primera condición.

Para la condición B, hay que tomar en cuenta que $L_1$ es el conjunto de todas las cadenas que codifican a los ejemplares-sí de 3-SAT mientras que $L_2$ es el conjunto de todas las cadenas que codifican a los ejemplares-sí de 3DM.

A continuación se presenta la demostración de que la existencia de una asignación de valores satisfactibles para el ejemplar de 3-SAT implica que $M$ contiene un emparejamiento [Garey & Johnson, 1979].

1. Se construye el subconjunto $T'$ (subconjunto de $M'$) del problema de 3DM de la siguiente manera:

$$T' = \{(\neg u_{i,j}, a_{i,j}, b_{i,j}) \rightarrow u_i = \text{True} \;||\; (u_{i,j}, a_{i,j}, b_{i,j}) \cup (u_{i,m}, a_{i,1}, b_{i,m}) \rightarrow u_i = \text{False} : 1 \le j \le m, 1 \le i \le n\}.$$

2. Se construye el subconjunto $C'$ (subconjunto de $M'$)del problema de 3DM de la siguiente manera:

   $C' = \{z_{j,j}, s_{1,j}, s_{2,j}\}$ donde $z_j = \{u_{i,j}, \neg u_{i,j}: 1 \leq i \leq n\}$ , $z_{j,j} \notin T'$ y $t(z_j)$ = True.

3. Se construye el subconjunto $G'$ (subconjunto de $M'$)del problema de 3DM de la siguiente manera:

   $$G' = \{(\neg u_{i,j}, u_{i,j}), g_{1,k}, g_{2,k} \mid (\neg u_{i,j}, u_{i,j}) \notin T' \wedge (\neg u_{i,j}, u_{i,j}) \notin C'\}$$

Para la verificación de la condición B1, considérese un ejemplar-sí $p$ de 3-SAT, entonces si $p$ es un ejemplar-sí debe haber una asignación de valores a las variables $u_i \in U$ que hagan verdadera a cada una de las cláusulas $c_i \in C$, y por lo tanto, al aplicar el algoritmo Trans 3-SAT-3DM la conformación del conjunto $M'$ del ejemplar $q$ de 3DM quedará incompleta. La selección del conjunto $T'$ asegura la aparición de todos los elementos de los subconjuntos $A \in X$ y $B \in Y$ sin que se repitan en cada una de las tripletas. De la misma forma, la selección del conjunto $C' \in M'$ asegura la elección de todos los elementos de los conjuntos $S_1 \in X$ y $S_2 \in Y$ sin que se repitan en ninguna tripleta. La selección del conjunto $G' \in M'$ asegura la aparición de todos los elementos del conjunto $G_1 \in X$ y $G_2 \in Y$ sin repetirse en ninguna de las tripletas. Debido a que $p$ es un ejemplar-sí, cada cláusula $c_i \in C$ tiene al menos una literal que hace verdadera a la cláusula, permitiendo completar el subconjunto $C' \in M'$ en el paso 9 de la transformación Trans 3-SAT-3DM, que en sí, es el paso que determina si el ejemplar $q$ será un ejemplar-sí o un ejemplar-no. Por ejemplo, considérese la cláusula $c_i = (u_1, u_2, u_3) \in C$ que pertenece al ejemplar $p$ de 3-SAT con los valores $u_1 = T$, $u_2 = F$ y $u_3 = F$, entonces, en la formación del conjunto $C' \in M'$, se seleccionará de $C \in M$ la tripleta $u_{1,j}, s_{1,j}, s_{2,j}$ debido a la condición $t(z_j)$ = True que indica que se seleccionará la tripleta con la literal $u_i$ que haga verdadera la cláusula $j$, en este caso $t(u_1)$ = True. Por otra parte, si la asignación de valores para $c_i$ es $u_1 = F$, $u_2 = F$ y $u_3 = F$, entonces en la formación del conjunto $C'$ no se podrá seleccionar ninguna tripleta de $C$ debido a que ninguna literal hace verdadera la cláusula, quedando incompleto el subconjunto $C'$ y por ende el conjunto $M'$, que estará formado por menos de $q$ elementos. Por lo tanto, la transformación Trans 3-SAT-3DM cumple la condición B1.

Para la verificación de la condición B2, se utilizará la prueba por contradicción; por lo tanto, supóngase que si $x \notin L_1$ entonces $f(x) \in L_2$. Si $x \notin L_1$, entonces esto implica que $x$ es una cadena-no de $\sum_1{}^*$, y con lo cual nos lleva a dos posibles casos: que $x$ sea una codificación de un ejemplar inválido de 3-SAT o que $x$ sea una codificación de un ejemplar-no de 3-SAT.

- Primer caso: Supóngase que $q$ fuera un ejemplar-sí de 3DM y que fuera la imagen de un ejemplar inválido $p$ de 3-SAT. El problema 3-SAT genera ejemplares inválidos cuando cualquiera de las cláusulas que conforman el ejemplar no está constituida por exactamente tres literales. Al aplicar el algoritmo de transformación a $p$, el paso 1 detectaría que se trata de un ejemplar inválido, en cuyo caso generaría un ejemplar inválido de 3DM y se detendría el proceso.

- Segundo caso: Supóngase que $q$ fuera un ejemplar-sí de 3DM y que fuera la imagen de un ejemplar-no $p$ de 3-SAT. El conjunto de ejemplares-no de 3-SAT está formado por aquellos ejemplares en los que la asignación de valores a las variables del ejemplar hacen la formula falsa; dicho de otra forma, son aquellos ejemplares que al momento de sustituir los valores en las cláusulas, al menos una de estas es falsa. Por lo tanto, para que una cláusula sea falsa, todas las literales deben adquirir el valor de falso. Supongamos que en la cláusula $c_j$, las tres literales toman un valor de False. En la fase de construcción de $M'$, por la condición $t(z_j)$ = True (selecciona de $C$ la tripleta con la literal $u_{i,j}$ que hace verdadera la cláusula $j$), no se seleccionará ninguna tripleta para el conjunto $C'$ debido a que todas las literales de la cláusula $c_j$ son falsas. El conjunto $M'$ quedará incompleto y consecuentemente, el supuesto de este caso es imposible.

### 5.4.3 Verificación de las condiciones implícitas

Con respecto a la primera condición C1, el pseudoalgoritmo toma como entrada cualquier ejemplar de 3-SAT (sea ejemplar-sí, ejemplar-no o ejemplar inválido). El algoritmo transforma los ejemplares-sí de 3-SAT a ejemplares-sí de 3DM. También transforma ejemplares-no de 3-SAT en ejemplares-no de 3DM.

Para verificar la condición C2 primero supongamos que tenemos un ejemplar $p$ de 3-SAT y un algoritmo $A_{3DM}$ que puede resolver cualquier ejemplar de 3DM (donde resolver un ejemplar significa responder "sí" o "no" para este ejemplar). Entonces, supongamos que obtenemos un ejemplar $q$ de 3DM al aplicar a $p$ el algoritmo de transformación y después utilizamos el algoritmo $A_{3DM}$ para resolver el ejemplar $q$. Como se ha comprobado que la transformación cumple la condición explícita B, si la respuesta del algoritmo $A_{3DM}$ es "*sí*" para el ejemplar $q$, entonces, por la condición B1' (si $f(x) \in L_2$ entonces $x \in L_1$) el ejemplar $p$ debe ser también un ejemplar-sí, de lo contrario, $p$ debe ser un ejemplar-no o inválido. Por lo tanto, la transformación cumple con la condición C2.

Con respecto a la condición implícita C3 no se ha podido demostrar la existencia de una transformación polinomial de 3DM a 3-SAT; de igual forma, de acuerdo con nuestra revisión bibliográfica, no se ha reportado dicha transformación en la literatura sobre la teoría de la completitud-NP.

### 5.4.4 Una peculiaridad de la transformación de 3-SAT a 3DM

Un hecho curioso de este algoritmo de transformación sucede cuando quitamos el paso uno del algoritmo de transformación Trans 3-SAT-3DM. Se encontró que al aplicar todo el algoritmo de transformación a ejemplares inválidos de 3-SAT (ejemplares en el que al menos una de sus cláusulas no tiene exactamente tres literales) se transformaban en ejemplares-sí de 3DM. Esto es debido a que la transformación de ejemplares-sí de 3-SAT a

ejemplares-sí de 3DM está en función de los valores booleanos que toman las variables del ejemplar de 3-SAT y no toma en cuenta el número de literales de cada cláusula.

Los componentes *truth-setting* y *fan-out* (paso 5) se forman en base a la asignación de valores booleanos de las variables; y por lo tanto, el número de tripletas que se formarán en este subconjunto de $M$ va a ser el mismo en ejemplares que tengan el mismo número de variables y de cláusulas. La formación de las tripletas del conjunto *satisfaction testing* (paso 6) se basa en las literales que se encuentran en cada cláusula, por lo cual, en el caso de un ejemplar de 3-SAT, se formarán tres tripletas por cada cláusula; sin embargo, no hay un límite en el número de tripletas a crear por cada cláusula, por lo cual, entre más literales tenga una cláusula, más tripletas contendrá el conjunto *satisfaction testing*. De la misma forma, la formación del subconjunto *garbage collection* (paso 7) se forma en base al número de variables y de cláusulas.

Ahora, en la conformación del conjunto $M'$ se seleccionan las tripletas del subconjunto $T'$ del conjunto *truth-setting* y *fan-out* (paso 9) en base al valor que tiene cada variable sin tomar en cuenta las cláusulas, por lo que un ejemplar con al menos una cláusula que no tenga exactamente tres literales no tendrá problemas en la selección de tripletas para este subconjunto de $M'$. En la selección de tripletas del conjunto *satisfaction testing* para el subconjunto $C'$ (paso 10), se toman en cuenta las literales que hacen verdadera cada cláusula; por lo tanto, no importa cuántas literales hay en cada cláusula del ejemplar de 3-SAT, sino que exista al menos una literal que tome el valor de verdad en cada cláusula, lo cual implica que no afecte a ningún ejemplar inválido de 3-SAT. En cuanto al proceso de selección de las tripletas del conjunto *garbage collection* para el conjunto $M$ (paso 11), tampoco toma en cuenta el número de literales de las cláusulas, por lo tanto, la selección no afectará el resultado del ejemplar.

Al estar en función de los valores booleanos de las variables, el algoritmo de transformación Trans 3-SAT-3DM garantiza transformar ejemplares-sí de 3-SAT a ejemplares-sí de 3DM y ejemplares-no de 3-SAT en ejemplares-no de 3DM, pero si se omitiera la verificación del número de literales de cada cláusula, permitiría también transformar cualquier ejemplar-sí de SAT a un ejemplar-sí de 3DM y cualquier ejemplar-no de 3-SAT a un ejemplar-no de 3DM.

Supongamos que tenemos un ejemplar *p* de SAT conformado de la siguiente manera: un conjunto de cláusulas $C$ = {($x_1$, ¬$x_2$, $x_3$, $x_4$), (¬$x_1$, ¬$x_2$, ¬$x_3$)} donde $x_1$=T, $x_2$=F, $x_3$=T, $x_4$. Se puede observar que la asignación de valores a las variables hace de $p$ un ejemplar-sí compuesto por cuatro variables ($n$) y dos cláusulas ($m$). Al aplicar el algoritmo Trans 3-SAT-3DM se obtendrá un ejemplar-sí $q$ de 3DM compuesto por los conjuntos $W$, $X$ y $Y$ con un tamaño cada uno de dieciséis ($2mn$) objetos. El resultado es un subconjunto $M'$ conformado por dieciséis tripletas sin ningún elemento repetido (Tabla 5.1).

Tabla 5.1.- Subconjunto $M'$ del ejemplar $q$ obtenido al aplicar el algoritmo de transformación Trans 3-SAT-3DM al ejemplar-sí $p$ de SAT.

| $T'$ | $C'$ | $G'$ |
|---|---|---|
| $\neg x_{1,1}, a_{1,1}, b_{1,1}$ | $x_{1,1}, s_{1,1}, s_{2,1}$ | $x_{1,1}, g_{1,1}, g_{2,1}$ |
| $\neg x_{1,2}, a_{1,2}, b_{1,2}$ | $\neg x_{2,2}, s_{1,2}, s_{2,2}$ | $\neg x_{2,1}, g_{1,2}, g_{2,2}$ |
| $x_{2,1}, a_{2,2}, b_{2,1}$ | | $x_{3,1}, g_{1,3}, g_{2,3}$ |
| $x_{2,2}, a_{2,1}, b_{2,2}$ | | $x_{3,2}, g_{1,4}, g_{2,4}$ |
| $\neg x_{3,1}, a_{3,1}, b_{3,1}$ | | $\neg x_{4,1}, g_{1,5}, g_{2,5}$ |
| $\neg x_{3,2}, a_{3,2}, b_{3,2}$ | | $\neg x_{4,2}, g_{1,6}, g_{2,6}$ |
| $x_{4,1}, a_{4,2}, b_{4,1}$ | | |
| $x_{4,2}, a_{4,1}, b_{4,2}$ | | |

Ahora, si tomamos de nueva cuenta el ejemplar $p$ de SAT y le quitamos la literal $x_3$ a su primera cláusula obtendremos un ejemplar de 3-SAT compuesto de la siguiente manera: un conjunto de cláusulas $C$ = {($x_1$, ¬$x_2$, $x_4$), (¬$x_1$, ¬$x_2$, ¬$x_3$)} donde $x_1$=$T$, $x_2$=$F$, $x_3$=$T$, $x_4$. Se puede observar que la asignación de valores a las variables hace de $p$ un ejemplar-sí compuesto por cuatro variables ($n$) y dos cláusulas ($m$). Si nos damos cuenta la única diferencia que existe con el ejemplar $p$ de SAT anterior es que ahora ambas cláusulas cuentan exactamente con tres literales y sigue manteniendo el mismo número de variables y la misma asignación de valores que hace a este ejemplar $p$ de 3-SAT un ejemplar-sí. Por lo tanto, al aplicar el algoritmo de transformación Trans 3-SAT-3DM se obtiene el mismo ejemplar-sí $q$ que con el ejemplar $p$ de SAT.

Esto representaría un gran problema si se intenta utilizar esta transformación para transformar un ejemplar $p$ de SAT en un ejemplar $q$ de 3DM, pues como pudimos observar, $q$ podría ser la imagen de más de un ejemplar en SAT.

## 5.5 La revisión de la transformación entre los problemas de SAT y 3-SAT

**SATISFACTIBILIDAD (SAT):** Dado un conjunto de variables booleanas $U$ = {$u_1$, $u_2$, …, $u_m$}, una colección de cláusulas booleanas $C$ = {$c_1$, $c_2$, ..., $c_m$} donde cada $c_j$ = {$z_1$, $z_2$, ..., $z_{k\,i}$} para $i$ = 1, …,$m$ (donde cada $z$ representa una variable $u$ o su negación ¬$u$, y es llamada *literal*) y una fórmula lógica $f$($u$) expresada en la forma normal conjuntiva (CNF) empleando las variables $U$ y las cláusulas $C$, entonces el problema consiste en determinar si existe una asignación de verdad $f$: $U \rightarrow$ [$T$, $F$], tal que $f$($u$) = $T$;

El siguiente algoritmo es una descripción resumida del proceso de transformación de 3-SAT a 3DM [Garey & Johnson, 1979].

### 5.5.1 Algoritmo Trans SAT-3-SAT:

0. Entrada: un ejemplar (sí, no o inválido) $p$ del problema 3-SAT.

1. Si el ejemplar es inválido (no reconoce simbología) generar un ejemplar inválido $q$ de 3-SAT y parar el proceso, en caso contrario, continuar con el paso 2.

2. Se define el siguiente conjunto de variables para 3-SAT:

$$U' = U \cup U'_1 \cup U'_2 \cup \ldots \cup U'_m$$

donde $U$ es el conjunto de variables de SAT y para cada cláusula $c_j$ de SAT:

$U'_j = \{y_{j,1}, y_{j,2}\}$, si $k$=1;
$U'_j = \{y_{j,1}\}$, si $k$=2;
$U'_j = \varnothing$, si $k$=3;
$U'_j = \{y_{j,i}: 1 \leq i \leq k-3\}$, si $k \geq 4$.

(Nota: $k$ es el número de literales en la cláusula $c_j$.)

3. Se define el siguiente conjunto de cláusulas para3-SAT:

$$C' = C'_1 \cup C'_2 \cup \ldots \cup C'_m$$

donde, por cada cláusula $c_j = \{z_1, z_2, \ldots, z_k\}$ de SAT:

*Caso 1:* $C'_j = \{(z_1, y_{j,1}, y_{j,2}), (z_1, y_{j,1}, \neg y_{j,2}), (z_1, \neg y_{j,1}, y_{j,2}), (z_1, \neg y_{j,1}, \neg y_{j,2})\}$, si $k$=1;
*Caso 2:* $C'_j = \{(z_1, z_2, y_{j,1}), (z_1, z_2, \neg y_{j,1})\}$, si $k$=2;
*Caso 3:* $C'_j = \{c_j\}$, si $k$=3;
*Caso 4:* $C'_j = \{(z_1, z_2, y_{j,1})\} \cup \{\{(\neg y_{j,i}, z_{i+2}, y_{j,i+1}): 1 \leq i \leq k-4\} \cup \{(\neg y_{j,k-3}, z_{k-1}, z_k)\}$, si $k \geq 4$.

### 5.5.2 Verificación de las condiciones explícitas

Para verificar la condición explícita A basta con observar el pseudoalgoritmo anterior para darse cuenta que todos los pasos tienen una complejidad polinomial, por lo cual se cumple la primera condición.

Para la condición B, hay que tomar en cuenta que $L_1$ es el conjunto de todas las cadenas que codifican a los ejemplares-sí de SAT mientras que $L_2$ es el conjunto de todas las cadenas que codifican a los ejemplares-sí de 3-SAT.

Para la verificación de la condición B1, considérese un ejemplar cualquiera $p$ de SAT, entonces si $p$ es un ejemplar-sí debe haber una asignación de valores a cada variable $u_i \in$

$U$ que haga verdadera la fórmula lógica $f$. Por lo anterior se deduce que para que la fórmula lógica obtenga un valor verdadero, al menos debe existir una variable $u_i$ en cada cláusula $c_j \in C$ tal que $f(u_i) = T$. La reducción asegura que, en caso de que una cláusula $c_j$ tenga una asignación de verdad, la asignación de verdad se mantenga al obtener el nuevo conjunto de cláusulas $C'$. Si la cláusula a transformar pertenece al caso uno o el caso dos, la combinación de las formas de las variables agregadas ($y_j^i$) con las variables originales $u_i \in c_j$ asegura que para las cláusulas obtenidas $c' \in C'$ la asignación de verdad se mantenga. En caso de que la cláusula pertenezca al caso tres, no sufre modificaciones, y por lo tanto, su asignación de verdad se mantiene. Si la cláusula pertenece al caso cuatro, se determinan los valores de las variables agregadas ($y_j^i$) dependiendo de qué variables originales hacen verdadera a la cláusula, lo cual hace que se mantenga la asignación de verdad.

Para la verificación de la condición B2, se utilizará la prueba por contradicción; por lo tanto, supóngase que si $x \notin L_1$ entonces $f(x) \in L_2$. Si $x \notin L_1$, entonces esto implica que $x$ es una cadena-no de $\sum_1{}^*$; y con lo cual nos lleva a dos posibles casos: que $x$ sea una codificación de un ejemplar inválido de SAT o que $x$ sea una codificación de un ejemplar-no de SAT.

- Primer caso: Supóngase que $q$ fuera un ejemplar-sí de 3-SAT y que fuera la imagen de un ejemplar inválido $p$ de SAT. El problema SAT genera ejemplares inválidos cuando la codificación del ejemplar no pertenece a la codificación de un ejemplar de SAT; entonces, al aplicar el algoritmo de transformación a $p$, el paso 1 detectaría que se trata de un ejemplar inválido, en cuyo caso generaría un ejemplar inválido de 3DM y se detendría el proceso.
- Segundo caso: Supóngase que $q$ fuera un ejemplar-sí de 3-SAT y que fuera la imagen de un ejemplar-no $p$ de SAT. El conjunto de ejemplares-no de SAT está formado por aquellos ejemplares en los que la asignación de valores a las variables del ejemplar hacen la fórmula falsa; dicho de otra forma, son aquellos ejemplares que al momento de sustituir los valores en las cláusulas, al menos una de éstas es falsa. Por lo tanto, para que una cláusula sea falsa, todas las literales deben adquirir el valor de falso. Supongamos que de la fórmula lógica $f$, alguna cláusula $c_j$ es falsa, esto haría a toda la fórmula falsa, y por lo tanto, $p$ sería un ejemplar-no. Al momento de transformar la cláusula $c_j$, primero se identifica a qué caso pertenece dependiendo del número de variables de la cláusula. Si la cláusula tuviera una o dos variables, las cláusulas obtenidas $C'$ seguirán siendo falsas debido a que en cada una de ellas se mantiene la variable original ($u_i$) que hace a la cláusula $c_j$ falsa. De la misma manera, si la cláusula $c_j$ tuviera tres variables, no sufriría ningún cambio y seguiría teniendo un valor de falso. Cuando la cláusula $c_j$ tiene más de tres variables, primero se necesita identificar cuál es la primera variable original ($u_i$) que haga la cláusula verdadera, con el fin de determinar los valores booleanos para las variables agregadas ($y_{j,i}$). Por lo anterior el supuesto de este caso es imposible.

### 5.5.3 Verificación de las condiciones implícitas

Con respecto a la primera condición C1, el pseudoalgoritmo toma como entrada cualquier ejemplar de SAT (sea ejemplar-sí, ejemplar-no o ejemplar inválido). El algoritmo transforma los ejemplares-sí de SAT a ejemplares-sí de 3-SAT. También transforma ejemplares-no de SAT en ejemplares-no de 3-SAT.

Para verificar la condición C2 primero supongamos que tenemos un ejemplar *p* de SAT y un algoritmo $A_{3\text{-SAT}}$ que puede resolver cualquier ejemplar de 3-SAT (donde resolver un ejemplar significa responder "sí" o "no" para este ejemplar). Entonces, supongamos que obtenemos un ejemplar $q$ de 3-SAT al aplicar a $p$ el algoritmo de transformación, y después utilizamos el algoritmo $A_{3\text{-SAT}}$ para resolver el ejemplar $q$. Como se ha comprobado que la transformación cumple la condición explícita B, si la respuesta del algoritmo $A_{3\text{-SAT}}$ es "*sí*" para el ejemplar $q$, entonces, por la condición B1' (si $f(x) \in L_2$ entonces $x \in L_1$) el ejemplar $p$ debe ser también un ejemplar-sí, de lo contrario, $p$ debe ser un ejemplar-no o inválido. Por lo tanto, la transformación cumple con la condición C2.

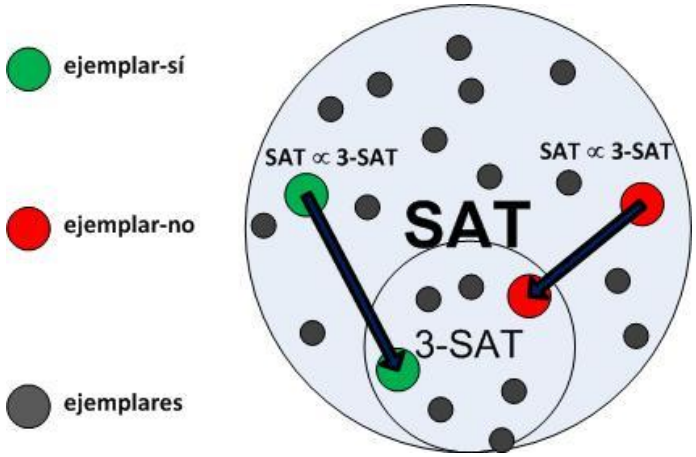



Figura 5.1.- La cardinalidad del problema SAT es mucho mayor que la cardinalidad del problema 3-SAT.

La verificación de la condición C3 se obtiene por la propia definición del problema SAT, dado que cada ejemplar del problema del 3-SAT es también un ejemplar del problema SAT (véase figura 5.1); es decir, aquél es un caso particular de éste; por lo tanto, la transformación en sentido inverso es simplemente la función identidad.

## 5.6 La revisión de la transformación entre los problemas de Knapsack y Partition

Uno de los pocos pares de problemas NP que se ha encontrado que existe una transformación polinomial en ambos sentidos es el de Knapsack y Partition, por lo que la

revisión de estas transformaciones es de vital importancia para tratar de conseguir los objetivos que busca este trabajo de tesis.

La transformación entre el problema de Knapsack y el problema de Partition es una de las primeras transformaciones realizadas dentro de la teoría de la completitud-NP. Fue una de las 21 transformaciones propuestas por [R. Karp, 1972].

**El problema Knapsack (mochila)**: Dados un conjunto finito de objetos $U$ y un tamaño $s(u) \in Z^+$ para cada objeto $u \in U$ y una mochila con capacidad $B \in Z^+$, el problema consiste en determinar si existe un subconjunto de objetos tal que la suma de sus tamaños ocupe la capacidad total de la mochila; es decir, si existe $U' \subset U$ tal que $\sum_{u \in U'} s(u) = B$.

El siguiente algoritmo es una representación resumida del proceso de transformación de Knapsack a Partition [R. Karp, 1972].

### 5.6.1 Algoritmo Trans Ks-Par:

0. Entrada: un ejemplar (sí, no o inválido) $p$ del problema Knapsack.

1. El número de elementos $n$ de Partition es igual al número de objetos $m$ de Knapsack más dos, es decir, $n = m + 2$.

2. Los primeros $m$ elementos de Partition son los objetos de Knapsack; es decir, $a_i = u_i$, y además, $s(a_i) = s(u_i)$, para $i = 1, 2, ..., m$.

3. El tamaño del primer elemento extra de Partition es igual a la sumatoria de los tamaños de los objetos de Knapsack más uno menos la capacidad de la mochila; es decir, $s(a_{m+1}) = \sum_{u \in U} s(u) + 1 - B$.

4. El tamaño del segundo elemento extra de Partition es igual a la capacidad $B$ de la mochila más uno; es decir, $s(a_{m+2}) = B + 1$.

### 5.6.2 Verificación de las condiciones explícitas

Para verificar la condición explícita A basta con observar el algoritmo Trans Ks-Par para darse cuenta que todos los pasos tienen una complejidad polinomial, por lo cual cumple la primera condición.

Para la condición explícita B, hay que tomar en cuenta que $L_1$ es el conjunto de todas las cadenas que codifican a los ejemplares-sí de Knapsack mientras que $L_2$ es el conjunto de todas las cadenas que codifican a los ejemplares-sí de Partition.

Considérese un ejemplar cualquiera $p$ de Knapsack, entonces si $p$ es un ejemplar-sí debe existir al menos un subconjunto $U'$ de $m'$ objetos tal que la suma de sus tamaños sea exactamente igual a la capacidad $B$ de la mochila, y por lo tanto, los dos elementos extras de Partition sirven de complemento para balancear los dos subconjuntos $A'$ (constituído por el primer elemento extra más los elementos correspondientes a los objetos de Knapsack que se almacenen en la mochila) y $A – A'$ (constituido por el segundo elemento extra más los elementos correspondientes a los objetos que no se almacenen en ella). Si existe un subconjunto de objetos $U'$ cuya suma de tamaños sea igual a la capacidad $B$ de la mochila, es decir $\sum_{u \in U'} s(u) = B$, entonces los dos elementos extras de Partition equilibran los pesos de los subconjuntos $A'$ y $A–A'$ en el ejemplar $p$ transformado, ya que por un lado $\sum_{a \in A'} s(a) = s(a_{m+1}) + \sum_{u \in U'} s(u) = \sum_{u \in U} s(u) + 1 – B + B = \sum_{u \in U} s(u) + 1$ y por otro lado $\sum_{a \in A-A'} s(a) = s(a_{m+2}) + \sum_{u \in U-U'} s(u) = B + 1 + \sum_{u \in U} s(u) – \sum_{u \in U'} s(u) = B + 1 + \sum_{u \in U} s(u) – B = \sum_{u \in U} s(u) + 1$; lo cual implica que $\sum_{a \in A'} s(a) = \sum_{a \in A-A'} s(a)$.

Para la verificación de la condición B2, se utilizará la prueba por contradicción; por lo tanto, supóngase que $x \notin L_1$ implica $f(x) \in L_2$. Si $x \notin L_1$, entonces esto implica que $x$ es una cadena-no de $\sum_1{}^*$, lo cual conduce a dos posibles casos: que $x$ sea una codificación de un ejemplar inválido de Knapsack o que $x$ sea una codificación de un ejemplar-no de Knapsack.

- Primer caso: Supóngase que $q$ fuera un ejemplar-sí de Partition y que fuera la imagen de un ejemplar inválido $p$ de Knapsack. El problema Knapsack no tiene muchas opciones para generar ejemplares inválidos; así que el que $p$ fuera inválido implicaría que el tamaño de alguno de sus objetos $s(u_i)$ no fuera un entero positivo; entonces, al aplicar el paso 2 del algoritmo Trans Ks-Par a $p$, resultaría que la imagen $q$ de $p$ tendría un elemento $a_i$ cuyo tamaño tampoco sería entero positivo, lo cual implicaría que $q$ fuera inválido, y por lo tanto, se contradice el supuesto de este caso. La otra posibilidad de que $p$ fuera inválido sería que la capacidad de la mochila $B$ no fuera un entero positivo; en este caso, al aplicar los pasos 3 y 4, ocurriría que los tamaños $s(a_{m+1})$ y $s(a_{m+2})$ de los elementos extras no serían enteros positivos, lo cual implicaría que $q$ fuera inválido, y por lo tanto, se contradice el supuesto de este caso.
- Segundo caso: Supóngase que $q$ fuera un ejemplar-sí de Partition y que fuera la imagen de un ejemplar-no de Knapsack. Existen dos subconjuntos de ejemplares-no de Knapsack: aquéllos para los que $\sum_{u \in U} s(u) < B$ (es decir, la capacidad $B$ de la mochila es mayor que la suma de los tamaños de todos los objetos del ejemplar $p$ de Knapsack), y aquéllos para los que $\sum_{u \in U} s(u) \geq B$ y para los que no es posible encontrar un subconjunto $U'$ de objetos de Knapsack tal que $\sum_{u \in U'} s(u) = B$, lo cual da lugar a dos subcasos:
  - Supóngase que para el ejemplar $p$ de Knapsack ocurre que $\sum_{u \in U} s(u) < B$; entonces, al aplicar el paso 4 del algoritmo Trans Ks-Par a $p$, resultaría que el tamaño del segundo elemento extra de Partition sería menor o igual a cero, lo cual implicaría que la imagen $q$ de $p$ fuera inválida, ya que violaría la condición de que todos los

tamaños de Partition sean enteros positivos; lo cual a su vez contradice la suposición de este caso.

- Supóngase que para el ejemplar $p$ de Knapsack sucede que $\sum_{u \in U} s(u) \geq B$ y no existe un subconjunto $U'$ de objetos tal que $\sum_{u \in U'} s(u) = B$, es decir para todo subconjunto $U'$ ocurre que $\sum_{u \in U'} s(u) \neq B$; por lo tanto, al aplicar la transformación a $p$ para obtener $q$, los dos elementos extras de $q$ no balancearían ningún par de subconjuntos de $A$; es decir, no sería posible encontrar dos subconjuntos $A'$ y $A–A'$ tales que $\sum_{a \in A'} s(a) = s(a_{m+1}) + \sum_{u \in U'} s(u) = \sum_{u \in U} s(u) + 1 - B + B = \sum_{u \in U} s(u) + 1$ y por otro lado $\sum_{a \in A-A'} s(a) = s(a_{m+2}) + \sum_{u \in U-U'} s(u) = B + 1 + \sum_{u \in U} s(u) - \sum_{u \in U'} s(u) = B + 1 + \sum_{u \in U} s(u) - B = \sum_{u \in U} s(u) + 1$.

### 5.6.3 Verificación de las condiciones implícitas

Con respecto a la primera condición C1, el algoritmo Trans Ks-Par toma como entrada cualquier ejemplar de Knapsack (sea ejemplar-sí, ejemplar-no o ejemplar inválido). El algoritmo transforma los ejemplares-sí de Knapsack a ejemplares-sí de Partition. También transforma ejemplares inválidos y ejemplares-no de Knapsack en ejemplares inválidos y ejemplares-no de Partition.

Para verificar la condición C2, supóngase que se tuviera un ejemplar $p$ de Knapsack y un algoritmo $A_{\text{Par}}$ capaz de resolver cualquier ejemplar de Partition (donde resolver un ejemplar significa contestar "sí" o "no" para éste), entonces nuestra tarea es diseñar un algoritmo $A_{\text{Ks}}$ basado en el algoritmo $A_{\text{Par}}$ que pueda resolver $p$. Para tal efecto considérese un algoritmo $A_{\text{Ks}}$ que consiste de dos fases: la primera usa el algoritmo Trans Ks-Par para transformar $p$ a un ejemplar $q$ de Partition, y la segunda fase usa el algoritmo $A_{\text{Par}}$ para resolver el ejemplar $q$. Dado que se ha demostrado que el algoritmo Trans Ks-Par satisface la condición explícita B, si la respuesta de $A_{\text{Par}}$ es "sí" para el ejemplar $q$, entonces la condición B1' (si $f(x) \in L_2$ entonces $x \in L_1$) implica que $p$ debe ser un ejemplar "sí"; en caso contrario $p$ debe ser un ejemplar "no" o un ejemplar inválido debido a la condición B2'. Por lo tanto, se ha demostrado que el algoritmo $A_{\text{Ks}}$ resuelve cualquier ejemplar de Knapsack.

Para la condición C3 se ha encontrado que existe una transformación revertida que transforma cualquier ejemplar de Partition en un ejemplar de Knapsack. A continuación se presenta esta transformación con la correspondiente verificación de condiciones explícitas e implícitas.

## 5.7 La revisión de la transformación entre los problemas de Partition y Knapsack

El siguiente algoritmo es una representación resumida del proceso de transformación de Partition a Knapsack [Tilburg University].

### 5.7.1 Algoritmo Trans Par-Ks1:

0. Entrada: un ejemplar (sí, no o inválido) $q$ del problema Partition.

1. El número de objetos $m$ de Knapsack es igual al número de elementos $n$ de Partition, es decir, $m = n$.

2. Para cada elemento $a_i$ de Partition definir un objeto $u_i$ de Knapsack tal que $s(u_i) = 2s(a_i)$.

3. Definir el valor de $B$ para el ejemplar de Knapsack tal que $B = \sum_{a \in A} s(a)$.

Con relación a este algoritmo, conviene aclarar que el tamaño de los objetos de Knapsack se tiene que hacer igual al doble del tamaño de los elementos de Partition para que la capacidad de la mochila sea un número entero; ya que si el tamaño de los objetos de Knapsack fuera igual al tamaño de los elementos de Partition, la capacidad de la mochila podría ser un número fraccionario en algunos casos. Sin embargo, esto puede acarrear problemas a la hora de transformar ciertos ejemplares inválidos, lo cual se verá más adelante en la revisión de esta transformación.

Para verificar la validez de esta transformación, primero se considerarán las condiciones explícitas A y B de la definición de transformación polinomial.

### 5.7.2 Verificación de las condiciones explícitas

Primeramente, como el algoritmo Trans Par-Ks1 tiene tres pasos cuya complejidad es polinomial, entonces el algoritmo satisface la condición A.

En segundo lugar, para la condición B hay que tomar en cuenta que $L_1$ es el conjunto de las cadenas que codifican a los ejemplares-sí de Partition mientras que $L_2$ es el de las cadenas que codifican a los ejemplares-sí de Knapsack.

En la verificación de la condición B1, primero considérese cualquier ejemplar-sí $q$ de Partition, en tales circunstancias los elementos de Partition deben poder dividirse en dos subconjuntos ($A'$ y $A–A'$) del mismo tamaño $2/\Sigma_{a \in A} s(a)$. Por otra parte, si $p$ es un ejemplar de Knapsack obtenido al aplicar a $q$ el algoritmo Trans Par-Ks1, entonces los objetos de Knapsack también deben poder dividirse en dos subconjuntos ($U'$ y $U–U'$, constituidos por los objetos correspondientes a los elementos de $A'$ y $A–A'$), cada uno con el mismo tamaño $2/\sum_{u \in U} s(u) = 2/\sum_{a \in A} 2s(a) = \sum_{a \in A} s(a) = B$. Lo anterior implica que el tamaño de cualquiera de los subconjuntos $U'$ y $U–U'$ sea igual a la capacidad de la mochila; y por lo tanto, $p$ debe ser un ejemplar-sí, lo cual finalmente implica que la condición B1 se satisfaga.

Para la verificación de la condición B2, se utilizará la prueba por contradicción; por lo tanto, supóngase que $x \notin L_1$ implica $f(x) \in L_2$. Si $x \notin L_1$, entonces esto implica que $x$ es una cadena-no de $\sum_1$*, lo cual conduce a dos posibles casos: que $x$ sea una codificación de un ejemplar inválido de Partition o que $x$ sea una codificación de un ejemplar-no de Partition.

- Primer caso: Para este caso no se utilizará la prueba por contradicción, ya que para este caso no se satisface la condición B2; en su lugar se utilizará un ejemplo para demostrar que no se satisface. Para tal efecto, considérese un ejemplar $q$ de Partition definido de la siguiente manera: un conjunto de elementos $\{a_1, a_2, a_3, a_4, a_5, a_6\}$ con sus respectivos tamaños $s(a_1) = 2.5$, $s(a_2) = 3.5$, $s(a_3) = 4.5$, $s(a_4) = 1.5$, $s(a_5) = 2.5$, $s(a_6) = 0.5$. Nótese que $q$ es inválido debido a que los tamaños de sus elementos no son enteros positivos. Al aplicar el algoritmo Trans Par-Ks1 al ejemplar $q$, se obtiene el ejemplar $p$, el cual consiste de un conjunto de objetos $U = \{u_1, u_2, u_3, u_4, u_5, u_6\}$ con sus respectivos tamaños $s(u_1) = 5$, $s(a_2) = 7$, $s(a_3) = 9$, $s(a_4) = 3$, $s(a_5) = 5$, $s(a_6) = 1$ y una capacidad $B = 15$ de la mochila. Obsérvese que $U' = \{u_1, u_2, u_4\}$ es un subconjunto de $U$ tal que $\sum_{u \in U'} s(u) = B$, lo cual implica que $p$ es un ejemplar-sí de Knapsack, y por lo tanto la condición B2 se viola. De lo anterior se puede concluir que la transformación definida por el algoritmo Trans Par-Ks1 es incorrecta.
- Segundo caso: Supóngase que $p$ fuera un ejemplar-sí de Knapsack y que fuera la imagen de un ejemplar-no $q$ de Partition. Existen dos tipos de ejemplares-no de Partition: aquéllos en los que $S = \sum_{a \in A} s(a)$ es impar y aquéllos en los que $S$ es par.
  - Si $\sum_{a \in A} s(a)$ fuera impar, esto implicaría que al aplicar el algoritmo Trans Par-Ks1 a $q$, resultaría que en la imagen $p$ de $q$ el tamaño de la mochila sería un número impar, pero los tamaños de los objetos son pares, lo cual haría imposible encontrar un subconjunto de objetos $U'$ de tamaño impar. Esto último implicaría que $p$ fuera un ejemplar-no de Knapsack, lo cual conduciría a una contradicción.
  - Si $\sum_{a \in A} s(a)$ fuera par, entonces como $q$ es un ejemplar-no, sucede que para cualquier posible partición de los elementos de Partition (en $A'$ y $A–A'$) debe suceder que $\sum_{a \in A} s(a) \neq S/2$ o $\sum_{a \in A\text{-}A'} s(a) \neq S/2$, lo cual implicaría que al aplicar el algoritmo Trans Par-Ks1 a $q$, resultaría que en la imagen $p$ de $q$ para cada posible subconjunto de objetos $U'$ de Knapsack sucedería que $\sum_{u \in U'} s(u) \neq B$. Esto último implicaría que $p$ fuera un ejemplar-no, lo cual daría lugar a una contradicción.

Como se pudo observar, la transformación cumple con las condiciones explícitas A y B1, pero en la condición B2, a pesar de que sí transforma todos los ejemplares-no de Partition en ejemplares-no de Knapsack, el problema es que a algunos ejemplares inválidos de Partition los transforma en ejemplares-sí de Knapsack.

Para prevenir la aparición de esta anomalía, se pueden realizar un par de correcciones en la transformación Trans Par-Ks1. A continuación se presenta la transformación Trans Par-Ks con las correcciones realizadas.

### 5.7.3 Algoritmo Trans Par-Ks2:

0. Entrada: un ejemplar (sí, no o inválido) $q$ del problema Partition.

1. El número de objetos $m$ de Knapsack es igual al número de elementos $n$ de Partition, es decir, $m = n$.

2. Para cada elemento $a_i$ de Partition definir un objeto $u_i$ de Knapsack tal que $s(u_i) = s(a_i)$.

3. Definir el valor de $B$ para el ejemplar de Knapsack tal que $B = 2\sum_{a \in A} s(a)$.

Se puede observar que en el paso 2 del algoritmo Trans Par-Ks2 se elimina la duplicación de los tamaños de los elementos de Partition, por lo que los objetos de Knapsack tendrán exactamente los mismos tamaños que los elementos de Partition ($s(u_i) = s(a_i)$).

La segunda corrección se presenta en el paso 3 del algoritmo Trans Par-Ks2, donde ahora la capacidad de la mochila ($B$) se hace igual a la mitad de la suma de todos los tamaños de los elementos del ejemplar de Partition ($B = 2\sum_{a \in A} s(a)$).

Para poder entender las modificaciones, supongamos que tenemos un ejemplar-sí $q$ de Partition. Al aplicar a $q$ el algoritmo Trans Par-Ks2 se obtendrá un ejemplar $p$ de Knapsack. El tamaño de cada objeto de $p$ será entero positivo debido al paso 2 del algoritmo ($s(u_i) = s(a_i)$) y la capacidad de la mochila $B$ también será de un valor entero positivo debido al paso 3 del algoritmo ($B = 2\sum_{a \in A} s(a)$), lo que convierte a $p$ en un ejemplar válido. Ahora, debido a que no hay modificaciones en los tamaños con respecto al ejemplar $q$, se sabe que existe un subconjunto de objetos cuya suma de tamaños es igual a $2\sum_{a \in A} s(a)$ que es la capacidad de la mochila, lo cual implica que $p$ sea un ejemplar-sí de Knapsack. En conclusión, el algoritmo Trans Par-Ks2 transforma todo ejemplar-sí de Partition en un ejemplar-sí de Knapsack, lo cual implica que Trans Par-Ks2 satisface la condición B1.

Ahora, supongamos que tenemos un ejemplar-no $q$ de Partition en el cual la suma de todos sus elementos fuera impar, entonces al aplicar el algoritmo Trans Par-Ks2 a $q$ se obtendrá un ejemplar inválido $p$ en Knapsack, debido a que en el paso 3 se obtendría un valor fraccionario para la capacidad $B$ de la mochila. Por ejemplo, considérese un ejemplar $q$ de Partition constituído por un conjunto de elementos $A = \{a_1, a_2, a_3, a_4\}$ con sus respectivos tamaños $s(a_1) = 2$, $s(a_2) = 3$, $s(a_3) = 1$, $s(a_4) = 1$. En este caso, ya que $\sum_{a \in A} s(a) = 7$ es un valor impar, resulta imposible formar dos subconjuntos $A'$ y $A–A'$ tales que $\sum_{a \in A'} s(a) = \sum_{a \in A\text{-}A'} s(a)$, lo cual implica que $q$ es un ejemplar-no. Si aplicamos el algoritmo Trans Par-Ks2 a $q$ se obtendrá un ejemplar $p$ de Knapsack conformado por los objetos $u_1 = 2$, $u_2 = 3$, $u_3 = 1$, $u_4 = 1$ y una capacidad de mochila $B = 2\sum_{a \in A} s(a) = 7/2 = 3.5$, lo cual implica que $q$ sea un ejemplar inválido. Con esto se demuestra que el algoritmo Trans Par-Ks2 satisface la condición B2 correspondiente al primer caso.

El hecho de que el algoritmo Trans Par-Ks2 transforme un ejemplar-no de Partition en un ejemplar invalido de Knapsack no viola la condición B2, ya que ésta exige que una cadena-no de un problema $\Pi_1$ (que corresponde a los ejemplares-no y a los inválidos de $\Pi_1$) sea transformada en una cadena-no de otro problema $\Pi_2$ (que corresponde a los ejemplares-no y a los inválidos de $\Pi_2$).

Una vez realizadas estas correcciones, podemos proceder a la revisión de las condiciones implícitas. La revisión de las condiciones explícitas (A y segundo caso de B2) para el algoritmo Par-Ks2 es similar a las del algoritmo Par-Ks1, y por lo tanto, se omitirán por limitación de espacio.

### 5.7.4 Verificación de las condiciones implícitas

Con respecto a la primera condición C1, el algoritmo Trans Par-Ks2 toma como entrada cualquier ejemplar de Partition (sea ejemplar-sí, ejemplar-no o ejemplar inválido). El algoritmo transforma los ejemplares-sí de Partition a ejemplares-sí de Knapsack. También transforma ejemplares inválidos y ejemplares-no de Partition en ejemplares inválidos y ejemplares-no de Knapsack.

Para verificar la condición C2 primero supóngase que se tuviera un ejemplar $q$ de Partition y un algoritmo $A_{\text{Ks}}$ que puede resolver cualquier ejemplar de Knapsack, entonces, nuestra tarea es diseñar un algoritmo $A_{\text{Par}}$ basado en el algoritmo $A_{\text{Ks}}$ que pueda resolver $q$. Para tal efecto considérese un algoritmo $A_{\text{Par}}$ que consiste de dos fases: la primera usa el algoritmo Trans Par-Ks2 para transformar $q$ a un ejemplar $p$ de Knapsack, y la segunda fase usa el algoritmo $A_{\text{Ks}}$ para resolver el ejemplar $p$. Dado que se ha demostrado que el algoritmo Trans Par-Ks2 satisface la condición explícita B, si la respuesta de $A_{\text{Ks}}$ es "sí" para el ejemplar $p$, entonces la condición B1' implica que $q$ debe ser un ejemplar "sí"; en caso contrario $q$ debe ser un ejemplar "no" o un ejemplar inválido debido a la condición B2'. Por lo tanto, se ha demostrado que el algoritmo $A_{\text{Par}}$ resuelve cualquier ejemplar de Partition.

Con respecto a la condición C3, se comprobó con el algoritmo Trans Par-Ks2 que la transformación puede realizarse en ambos sentidos.

### 5.7.5 Peculiaridades de las Transformaciones entre Knapsack y Partition

Los resultados de la revisión de la transformación entre el problema de Knapsack y el problema de Partition revelan un hecho destacable: en general la transformación revertida no obtiene la misma imagen ($p$) del ejemplar original de Partition, sino otro ejemplar ($r$), como se ilustra en la figura 5.2.

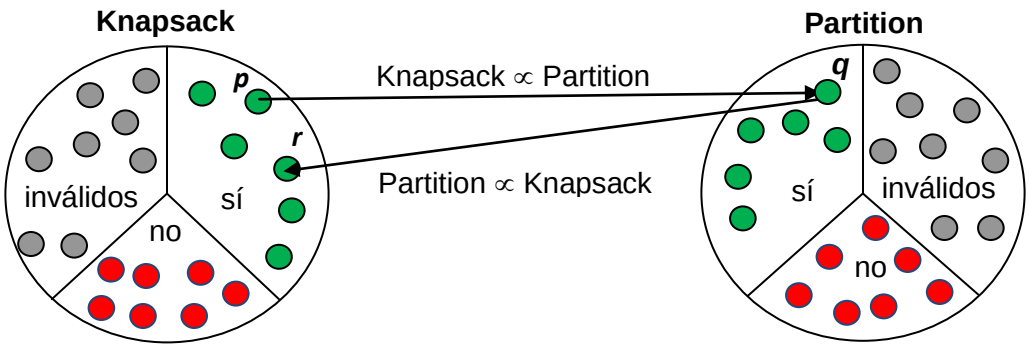



Figura 5.2.- Relación entre las transformaciones Knapsack ∝ Partition y Partition ∝ Knapsack.

La transformación revertida no obtiene el ejemplar equivalente de Partition, debido a que la transformación Trans Par-Ks2 no es la función inversa de Trans Ks-Par. Para ilustrar este fenómeno considérese un ejemplar $p$ de Knapsack definido de la siguiente manera: un conjunto de objetos $\{u_1, u_2, u_3\}$ con sus respectivos tamaños $s(u_1) = 3$, $s(u_2) = 2$, $s(u_3) = 5$ y una capacidad $B = 5$ de la mochila. La aplicación del algoritmo Trans Ks-Par produce un ejemplar $q$ de Partition definido como sigue: un conjunto de elementos $\{a_1, a_2, a_3, a_4, a_5\}$ con sus respectivos tamaños $s(a_1) = 3$, $s(a_2) = 2$, $s(a_3) = 5$, $s(a_4) = 6$, $s(a_5) = 6$. Si se aplica el algoritmo Trans Par-Ks2 a $q$, éste genera un ejemplar $r$ definido de la siguiente manera: un conjunto de objetos $\{u_1, u_2, u_3, u_4, u_5\}$ con sus respectivos tamaños $s(u_1) = 3$, $s(u_2) = 2$, $s(u_3) = 5$, $s(u_4) = 6$, $s(u_5) = 6$ y una capacidad $B = 11$ de la mochila. Obsérvese que el ejemplar $r$ tiene dos objetos adicionales a los del ejemplar original $p$ y la capacidad de la mochila es más grande.

Como se puede observar en la figura 5.3, los ejemplares-sí de Knapsack obtenidos por el algoritmo Trans Par-Ks2 también pueden verse como ejemplares-sí de Partition, dado que los objetos de estos ejemplares-sí de Knapsack pueden dividirse en dos subconjuntos de objetos cuya suma de tamaños es igual. Podría concluirse entonces que Partition es un caso particular de Knapsack, el cual corresponde a aquellos ejemplares de Knapsack en los que el tamaño acumulado de los objetos que son asignados a la mochila es igual al tamaño acumulado de los objetos que no son asignados. Sin embargo, hay ejemplares-sí de Knapsack que no pueden ser obtenidos de ejemplares-sí de Partition mediante el algoritmo Trans Par-Ks2; por ejemplo, considérese el ejemplar-sí de Knapsack definido de la siguiente manera: un conjunto de objetos $\{u_1, u_2, u_3\}$ con sus respectivos tamaños $s(u_1) = 3$, $s(u_2) = 2$, $s(u_3) = 4$ y una capacidad $B = 5$ de la mochila. Este ejemplar no es la imagen de ningún ejemplar-sí de Partition, ya que no es posible dividir el conjunto de objetos en dos subconjuntos cuya suma de tamaños sea igual. Esto implica que la cardinalidad del conjunto de ejemplares-sí de Knapsack es mayor que la cardinalidad del conjunto de los ejemplares-sí de Partition.

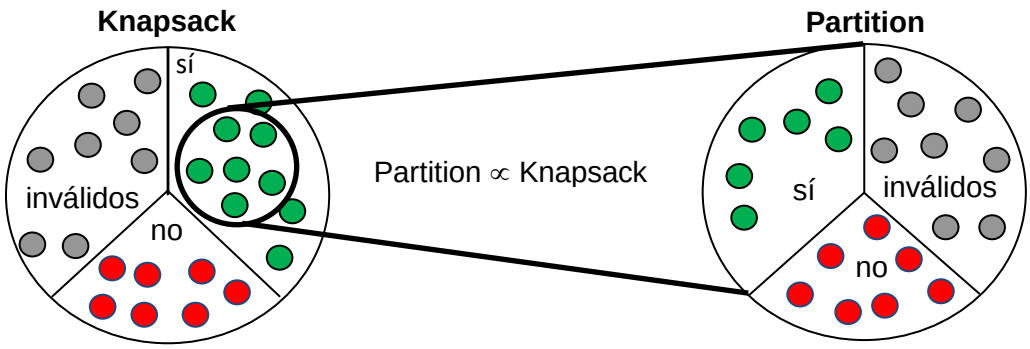



Figura 5.3.- Diferencia en la cardinalidad de los conjuntos de ejemplares-sí de Knapsack y Partition.

Otro hecho interesante que se desprende de los resultados de la revisión de estos problemas es el siguiente: dado que la cardinalidad del conjunto de los ejemplares-sí de Knapsack es mayor que la del conjunto de ejemplares-sí de Partition y el algoritmo Trans Kp-Par transforma todos los ejemplares-sí de Knapsack en ejemplares-sí de Partition, entonces el algoritmo Trans Kp-Par tiene la propiedad de transformar varios ejemplares-sí de Knapsack en un ejemplar-sí de Partition.

# Capítulo 6. Conclusiones y trabajos futuros

Se ha mostrado mediante la revisión de la transformación de Partition a Bin-Packing la dificultad de encontrar una transformación en sentido contrario (de Bin-Packing a Partition). Se han realizado cuatro diferentes intentos de transformación invertida, y ninguna llegó a cumplir todas las condiciones (explícitas e implícitas).

De esta transformación surgió la hipótesis de la no pertenencia del problema de Partition a la clase NP-completa y por este motivo se decidió revisar la rama de transformaciones que van desde el problema de Partition hasta el problema del SAT (3DM $\propto$ Partition, 3-SAT $\propto$ 3DM and SAT $\propto$ 3-SAT). Las revisiones a estas transformaciones revelaron que cada una de ellas cumple con las condiciones explícitas A, B1 y B2 y de la misma forma cumplen las condiciones implícitas C1 y C2; por lo tanto, concluimos que esta serie de transformaciones son correctas y consecuentemente se desecha la hipótesis de la no pertenencia a la clase NP-completo del problema de Partition.

Sin embargo, exceptuando la transformación polinomial SAT $\propto$ 3-SAT, las demás transformaciones incumplieron la condición explícita C3. Es importante aclarar, que a pesar de no poder haber demostrado que la condición C3 (la existencia de una transformación revertida) fuera satisfecha, esto no quiere decir que la transformación revertida de cualquiera de estas transformaciones no exista, sino que es demasiado difícil llegar a diseñar dicha transformación.

Otras de las revisiones realizadas fue la de la transformación Knapsack $\propto$ Partition, que al igual que la transformación 3DM $\propto$ Partition, comprueban la pertenencia del problema Partition a la clase NP-completa. Se comprobó que la transformación cumple con todas las condiciones (explícitas e implícitas), y además, de esta revisión se pudieron obtener las siguientes conclusiones:

- La transformación revertida aplicada sobre un ejemplar-sí $q$ de Partition no obtiene el mismo ejemplar-sí $p$ de Knapsack utilizado para obtener el ejemplar $q$ de Partition.
- La cardinalidad del conjunto de ejemplares-sí de Knapsack es mayor que la del conjunto de ejemplares-sí de Partition.
- Partition es un caso particular de Knapsack, en el cual, para un ejemplar-sí, el tamaño acumulado de los objetos asignados a la mochila es igual al de los objetos no asignados.

La transformación revertida Trans Par-Ks1 dentro de esta revisión, es un ejemplo claro de la necesidad de revisar el cumplimiento de las condiciones explícitas e implícitas para cada transformación.

Uno de los aspectos a destacar dentro de la revisión de la transformación Knapsack $\propto$ Partition es que nos muestra que la transformación revertida no obtiene la misma imagen de la transformación original. Lo más probable es que este fenómeno se presente en toda transformación revertida debido a la independencia de la transformación original.

Una de las suposiciones a tratar de comprobar es que no para cualquier par de problemas NP-completos existe una transformación polinomial revertida. A pesar de los intentos realizados, no se pudo comprobar o rechazar esta hipótesis.

Tal vez para poder encontrar una respuesta a esta anomalía, se requiera un análisis a fondo de los fundamentos de la teoría de la completitud-NP; concretamente, el trabajo de Cook-Levin en donde se introducen las primeras nociones de lo que hoy conocemos como teoría de la completitud-NP. Específicamente, la completitud del problema SAT y la definición de transformación polinomial (reducible-P como lo llama en su artículo) en la que establece que debe ser una función transitiva y equivalente.

Uno de los trabajos de importancia para un futuro es la implementación del teorema de Cook que establece que cualquier problema NP puede reducirse al problema SAT. La implementación de una máquina de Turing no determinista (MTND) con un tiempo polinomial que transforme un determinado problema NP al problema SAT bajo las especificaciones establecidas en el teorema de Cook seria de gran ayuda a la comprensión del mismo. En la época en la que se desarrolló el teorema de Cook era prácticamente imposible realizar la implementación de una máquina de Turing debido a la gran cantidad de recurso que requeriría; sin embargo, en la actualidad, con los avances en la tecnología hay grandes posibilidades de poder llegar a implementar la transformación descrita en dicho teorema.

En resumen, entre los trabajos futuros para este tema se proponen los siguientes:

- Revisión a fondo del teorema de Cook-Levin.
- Implementación de una MTND utilizada para transformar ejemplares de un problema NP al problema SAT.
- Intentar obtener transformaciones revertidas entre problemas NP-completos para los cuales aún no se conozca alguna.
- Realizar transformaciones polinomiales entre cualquier par de problemas NP-completos entre los cuales no exista una transformación polinomial aún en la literatura.
- Desarrollar transformaciones entra cualquier problema NP-completo al problema SAT bajo el método implícito utilizado en la demostración del teorema de Cook.

# Anexo A. Máquina de Turing determinista

En este anexo se muestra un ejemplo del diseño de una Máquina de Turing Determinista (MTD) aplicado a un problema en particular con el fin de comprender mejor el funcionamiento de estas máquinas.

Tomando como base las definiciones dadas en [Garey & Johnson, 1979] de una máquina de Turing determinista, se desarrolló un ejemplo de una MTD aplicada a un problema de manera detallada con un fin didáctico, en el cual se pueda explicar el funcionamiento de estas máquinas. El comprender el funcionamiento de una MTD es de vital importancia para el desarrollo de este trabajo; sin embargo, los ejemplos encontrados en la literatura carecían de profundidad y sólo ejemplificaban de manera muy superficial mediante algoritmos muy generales la aplicación de estas máquinas a problemas de decisión.

## A.1 Ejemplo de una máquina de Turing determinista

Para el entendimiento del siguiente ejemplo es necesario tener claros los conceptos de autómata finito determinista y máquina de Turing determinista (véase sección 2.3).

Para el ejemplo de una máquina de Turing no determinista se utilizó el problema de decisión de la suma de dos números binarios de dos dígitos, en el cual se busca dar respuesta a la pregunta siguiente: ¿La suma de dos números binarios de dos dígitos es menor o igual a un determinado número binario de tres dígitos?

Consideremos el siguiente ejemplar: ¿10 + 11 ≤ 110?

Se codifica el ejemplar en la cinta de la siguiente manera: 10, 11, 110$b$.

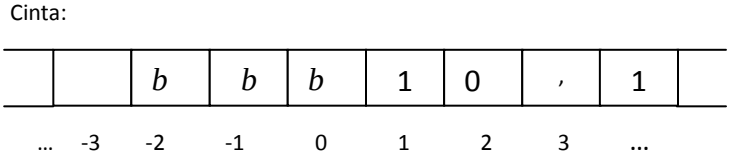



Figura A.1.- Representación codificada del ejemplar en la cinta.

En la figura A.1 se puede observar la representación codificada del ejemplar en la cinta, donde de las celdas desde la cero hacia la izquierda contienen símbolos blancos, a partir de la celda uno en adelante contiene los símbolos del ejemplar a evaluar, y por último, la celda final contiene un símbolo blanco que tiene la función de delimitador del ejemplar.

En la figura A.2 se muestra el autómata finito determinista (AFD) de la MTD.

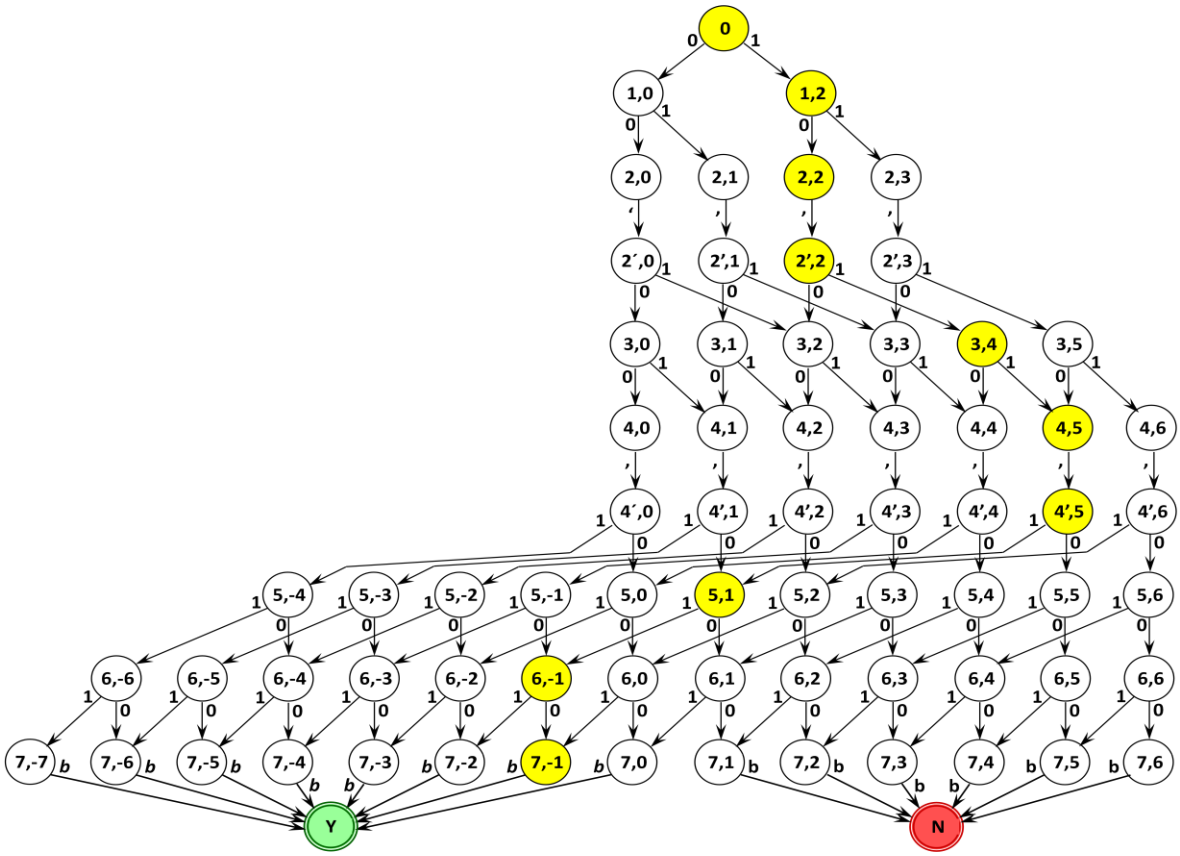



Figura A.2.- Autómata finito determinista de la máquina de Turing.

Este AFD está diseñado de tal manera que la etiqueta de cada estado lleve el cálculo del valor numérico de los símbolos que se hayan leído hasta el momento de llegar al estado; específicamente, mediante una etiqueta del estado que indica el paso en el que se encuentra el programa y el valor acumulado de los números binarios. Nota: para cada estado, la etiqueta indica el subíndice de la letra "$q$" que usualmente se usa para representar los estados; por ejemplo, la etiqueta 1,0 indica el estado $q_{1,0}$.

Para determinar si el tercer número es mayor o igual a la suma de los dos primeros, se realiza una resta, de tal manera que si al restarle a la suma de los dos primeros números el valor del tercer número nos da un número negativo o cero, entonces sí se cumple la condición y se considera como un ejemplar-sí. En caso contrario, no se cumple la condición y es un ejemplar-no.

Para los dos primeros números, las etiquetas de los estados indican la suma de los valores de los números binarios. Por ejemplo, si el primer número es "11" pasaría primero por el estado $q_{1,2}$, dado que el valor del primer "1" en binario es dos, y después pasaría por el

estado $q_{2,3}$, dado que el valor binario del segundo "1" es uno y se suma al dos que llevábamos en el subíndice del estado anterior.

Para realizar la resta el proceso es similar al de la suma, solamente que en lugar de sumar el valor binario del símbolo leído, se resta. Por ejemplo, si al final de la suma de los dos números binarios en el AFD nos encontramos en el estado $q_{4,4}$ y el primer símbolo del tercer número binario es "1", entonces el siguiente estado sería $q_{5,0}$, debido a que el símbolo "1" toma el valor de cuatro en el tercer número binario debido a que se encuentra en la posición más significativa; y por lo tanto, se le resta al valor del subíndice anterior, que era de cuatro.

Y al final, cuando un estado en el paso siete termina con un valor negativo o nulo en la etiqueta, la MTD termina en el estado de aceptación $q_Y$, de lo contrario termina en el estado de rechazo $q_N$.

Cabe destacar que, si la MTD recibe un símbolo no esperado para la celda en la que se encuentra, entonces termina en un estado inválido. En el AFD se suprimieron las transiciones que llevan a los estados inválidos "$q_I$" debido a que abarcarían mucho espacio y serian demasiados estados y transiciones. Por ejemplo, si en el estado $q_{1,2}$, el siguiente símbolo leído es una ",", esto nos llevaría en la función de transición al estado $q_I$, un estado invalido, puesto que se esperaba en esa celda un símbolo "1" o "0".

En este ejemplo podemos definir nuestros conjuntos base:

Conjunto de símbolos del alfabeto $\sum$ = {0, 1, ","}.

Conjunto de símbolos de entrada $\Gamma$ = {0, 1, ",", $b$}.

Conjunto de estados finitos $Q$ = {$q_0, q_{0,1}, \ldots, q_Y, q_N, q_I$}.

Una función de transición $\delta$= {$q \in Q, s \in \Gamma$}.

Y la tabla de transición de estados quedaría de la siguiente manera:

Tabla A.1.- Tabla de transición de estados.

| $q$ | 0 | 1 | "," | $b$ |
|---|---|---|---|---|
| $q_0$ | ($q_{1,0}$, 0, +1) | ($q_{1,2}$, 1, +1) | $q_I$ | $q_I$ |
| $q_{1,0}$ | ($q_{2,0}$, 0, +1) | ($q_{2,1}$, 1, +1) | $q_I$ | $q_I$ |
| $q_{1,2}$ | ($q_{2,2}$, 0, +1) | ($q_{2,3}$, 1, +1) | $q_I$ | $q_I$ |

| | | | | |
|---|---|---|---|---|
| $q_{2,0}$ | $(q_{3,0}, 0, +1)$ | $(q_{3,2}, 1, +1)$ | $q_I$ | $q_I$ |
| $q_{2,1}$ | $(q_{3,1}, 0, +1)$ | $(q_{3,3}, 1, +1)$ | $q_I$ | $q_I$ |
| $q_{2,2}$ | $(q_{3,2}, 0, +1)$ | $(q_{3,4}, 1, +1)$ | $q_I$ | $q_I$ |
| $q_{2,3}$ | $(q_{3,3}, 0, +1)$ | $(q_{3,5}, 1, +1)$ | $q_I$ | $q_I$ |
| $q_{3,0}$ | $(q_{4,0}, 0, +1)$ | $(q_{4,1}, 1, +1)$ | $q_I$ | $q_I$ |
| $q_{3,1}$ | $(q_{4,1}, 0, +1)$ | $(q_{4,2}, 1, +1)$ | $q_I$ | $q_I$ |
| $q_{3,2}$ | $(q_{4,2}, 0, +1)$ | $(q_{4,3}, 1, +1)$ | $q_I$ | $q_I$ |
| $q_{3,3}$ | $(q_{4,3}, 0, +1)$ | $(q_{4,4}, 1, +1)$ | $q_I$ | $q_I$ |
| $q_{3,4}$ | $(q_{4,4}, 0, +1)$ | $(q_{4,5}, 1, +1)$ | $q_I$ | $q_I$ |
| $q_{3,5}$ | $(q_{4,5}, 0, +1)$ | $(q_{4,6}, 1, +1)$ | $q_I$ | $q_I$ |
| $q_{4,0}$ | $(q_{5,0}, 0, +1)$ | $(q_{5,-4}, 1, +1)$ | $q_I$ | $q_I$ |
| $q_{4,1}$ | $(q_{5,1}, 0, +1)$ | $(q_{5,-3}, 1, +1)$ | $q_I$ | $q_I$ |
| $q_{4,2}$ | $(q_{5,2}, 0, +1)$ | $(q_{5,-2}, 1, +1)$ | $q_I$ | $q_I$ |
| $q_{4,3}$ | $(q_{5,3}, 0, +1)$ | $(q_{5,-1}, 1, +1)$ | $q_I$ | $q_I$ |
| $q_{4,4}$ | $(q_{5,4}, 0, +1)$ | $(q_{5,0}, 1, +1)$ | $q_I$ | $q_I$ |
| $q_{4,5}$ | $(q_{5,5}, 0, +1)$ | $(q_{5,1}, 1, +1)$ | $q_I$ | $q_I$ |
| $q_{4,6}$ | $(q_{5,6}, 0, +1)$ | $(q_{5,2}, 1, +1)$ | $q_I$ | $q_I$ |
| $q_{4',0}$ | $q_I$ | $q_I$ | $(q_{4,0}, ",", +1)$ | $q_I$ |
| $q_{4',1}$ | $q_I$ | $q_I$ | $(q_{4,1}, ",", +1)$ | $q_I$ |
| $q_{4',2}$ | $q_I$ | $q_I$ | $(q_{4,2}, ",", +1)$ | $q_I$ |
| $q_{4,'3}$ | $q_I$ | $q_I$ | $(q_{4,3}, ",", +1)$ | $q_I$ |

| | | | | |
|---|---|---|---|---|
| $q_{4',4}$ | $q_I$ | $q_I$ | $(q_{4,4}, ",", +1)$ | $q_I$ |
| $q_{4',5}$ | $q_I$ | $q_I$ | $(q_{4,5}, ",", +1)$ | $q_I$ |
| $q_{4',6}$ | $q_I$ | $q_I$ | $(q_{4,6}, ",", +1)$ | $q_I$ |
| $q_{5,-4}$ | $(q_{6,-4}, 0, +1)$ | $(q_{6,-6}, 1, +1)$ | $q_I$ | $q_I$ |
| $q_{5,-3}$ | $(q_{6,-3}, 0, +1)$ | $(q_{6,-5}, 1, +1)$ | $q_I$ | $q_I$ |
| $q_{5,-2}$ | $(q_{6,-2}, 0, +1)$ | $(q_{6,-4}, 1, +1)$ | $q_I$ | $q_I$ |
| $q_{5,-1}$ | $(q_{6,-1}, 0, +1)$ | $(q_{6,-3}, 1, +1)$ | $q_I$ | $q_I$ |
| $q_{5,0}$ | $(q_{6,0}, 0, +1)$ | $(q_{6,-2}, 1, +1)$ | $q_I$ | $q_I$ |
| $q_{5,1}$ | $(q_{6,1}, 0, +1)$ | $(q_{6,-1}, 1, +1)$ | $q_I$ | $q_I$ |
| $q_{5,2}$ | $(q_{6,2}, 0, +1)$ | $(q_{6,0}, 1, +1)$ | $q_I$ | $q_I$ |
| $q_{5,3}$ | $(q_{6,3}, 0, +1)$ | $(q_{6,1}, 1, +1)$ | $q_I$ | $q_I$ |
| $q_{5,4}$ | $(q_{6,4}, 0, +1)$ | $(q_{6,2}, 1, +1)$ | $q_I$ | $q_I$ |
| $q_{5,5}$ | $(q_{6,5}, 0, +1)$ | $(q_{6,3}, 1, +1)$ | $q_I$ | $q_I$ |
| $q_{5,6}$ | $(q_{6,6}, 0, +1)$ | $(q_{6,4}, 1, +1)$ | $q_I$ | $q_I$ |
| $q_{6,-6}$ | $(q_{7,-6}, 0, +1)$ | $(q_{7,-7}, 1, +1)$ | $q_I$ | $q_I$ |
| $q_{6,-5}$ | $(q_{7,-5}, 0, +1)$ | $(q_{7,-6}, 1, +1)$ | $q_I$ | $q_I$ |
| $q_{6,-4}$ | $(q_{7,-4}, 0, +1)$ | $(q_{7,-5}, 1, +1)$ | $q_I$ | $q_I$ |
| $q_{6,-3}$ | $(q_{7,-3}, 0, +1)$ | $(q_{7,-4}, 1, +1)$ | $q_I$ | $q_I$ |
| $q_{6,-2}$ | $(q_{7,-2}, 0, +1)$ | $(q_{7,-3}, 1, +1)$ | $q_I$ | $q_I$ |
| $q_{6,-1}$ | $(q_{7,-1}, 0, +1)$ | $(q_{7,-2}, 1, +1)$ | $q_I$ | $q_I$ |
| $q_{6,0}$ | $(q_{7,0}, 0, +1)$ | $(q_{7,-1}, 1, +1)$ | $q_I$ | $q_I$ |

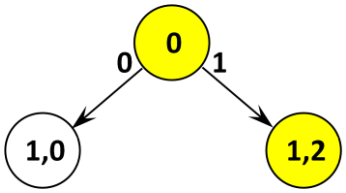
| | | | | |
|---|---|---|---|---|
| $q_{6,1}$ | $(q_{7,1}, 0, +1)$ | $(q_{7,0}, 1, +1)$ | $q_I$ | $q_I$ |
| $q_{6,2}$ | $(q_{7,2}, 0, +1)$ | $(q_{7,1}, 1, +1)$ | $q_I$ | $q_I$ |
| $q_{6,3}$ | $(q_{7,3}, 0, +1)$ | $(q_{7,2}, 1, +1)$ | $q_I$ | $q_I$ |
| $q_{6,4}$ | $(q_{7,4}, 0, +1)$ | $(q_{7,3}, 1, +1)$ | $q_I$ | $q_I$ |
| $q_{6,5}$ | $(q_{7,5}, 0, +1)$ | $(q_{7,4}, 1, +1)$ | $q_I$ | $q_I$ |
| $q_{6,6}$ | $(q_{7,6}, 0, +1)$ | $(q_{7,5}, 1, +1)$ | $q_I$ | $q_I$ |
| $q_{7,-7}$ | $q_I$ | $q_I$ | $q_I$ | $q_Y$ |
| $q_{7,-6}$ | $q_I$ | $q_I$ | $q_I$ | $q_Y$ |
| $q_{7,-5}$ | $q_I$ | $q_I$ | $q_I$ | $q_Y$ |
| $q_{7,-4}$ | $q_I$ | $q_I$ | $q_I$ | $q_Y$ |
| $q_{7,-3}$ | $q_I$ | $q_I$ | $q_I$ | $q_Y$ |
| $q_{7,-2}$ | $q_I$ | $q_I$ | $q_I$ | $q_Y$ |
| $q_{7,-1}$ | $q_I$ | $q_I$ | $q_I$ | $q_Y$ |
| $q_{7,0}$ | $q_I$ | $q_I$ | $q_I$ | $q_Y$ |
| $q_{7,1}$ | $q_I$ | $q_I$ | $q_I$ | $q_N$ |
| $q_{7,2}$ | $q_I$ | $q_I$ | $q_I$ | $q_N$ |
| $q_{7,3}$ | $q_I$ | $q_I$ | $q_I$ | $q_N$ |
| $q_{7,4}$ | $q_I$ | $q_I$ | $q_I$ | $q_N$ |
| $q_{7,5}$ | $q_I$ | $q_I$ | $q_I$ | $q_N$ |
| $q_{7,6}$ | $q_I$ | $q_I$ | $q_I$ | $q_N$ |

donde cada transición está formada por una triada de estado, símbolo y movimiento. El estado indica a qué estado se dirige, el símbolo es el que escribirá en la celda actual y el movimiento se representa con +1 si la cabeza se mueve a la derecha, -1 a la izquierda y 0 si no se mueve.

Ahora revisaremos cada uno de los pasos que sigue la MTD para determinar si el ejemplar propuesto al inicio es un ejemplar-sí o es un ejemplar-no.

Al inicio del proceso de la MTD, la cabeza se posiciona en la celda uno, para iniciar el proceso de transición de estados.

La cabeza lee el símbolo correspondiente a la celda uno, después transfiere el símbolo leído al CEF para determinar la transición correspondiente. Como el primer símbolo leído es "1" y nos encontramos en el estado inicial $q_0$, entonces busca la transición para el símbolo "1" y el estado inicial $q_0$.




| $q$ | 0 | 1 | "," | $b$ |
|---|---|---|---|---|
| $q_0$ | $(q_{1,0}, 0, +1)$ | $(q_{1,2}, 1, +1)$ | $q_I$ | $q_I$ |

Figura A.3.- Función de transición del primer símbolo.

La función de transición para el símbolo "1" nos lleva al estado $q_{1,2}$, (figura A.3) y manda una orden a la cabeza para que escriba el símbolo uno y que se mueva una celda a la derecha. En la representación del estado, el primer número del subíndice indica el paso en el que se encuentra la MTD y el segundo número del subíndice indica el valor que lleva la suma binaria. En este paso, como el primer símbolo es "1", entonces su valor binario es de dos por encontrarse en la posición más significativa.

Después de cambiar de estado, la MTD se mueve una celda a la derecha para leer el siguiente símbolo, el cual es "0". La función de transición para este estado nos lleva al estado $q_{2,2}$, (figura A.4) que indica que la MTD se encuentra en el paso dos con una suma de dos, debido a que el símbolo leído "0" en la posición menos significativa tiene el valor de cero, y esto se le suma a la suma que llevábamos, que era de dos. Enseguida la cabeza realiza un movimiento de una celda a la derecha.

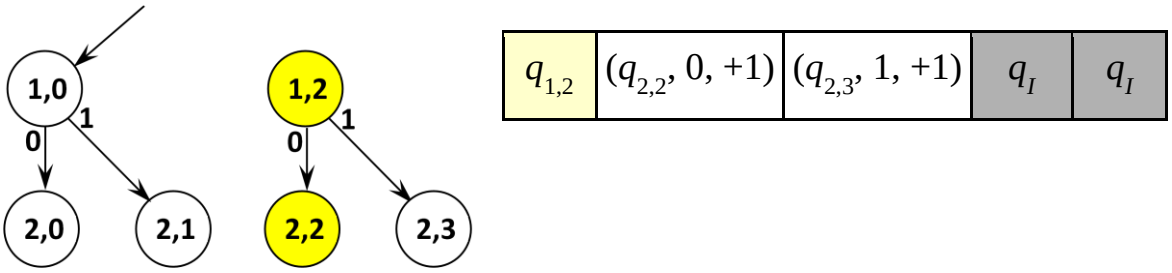



Figura A.4.- Función de transición para el segundo símbolo.

En la tercera casilla leerá el separador "," que separa los valores binarios. La función de transición para este símbolo se muestra en la figura A.5.

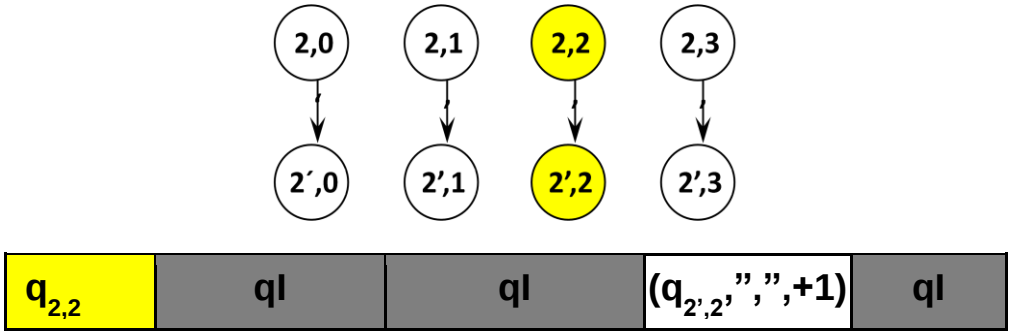



Figura A.5.- Función de transición para el símbolo ",".

Ahora el CEF pasa al estado $q_{2',2,}$ que indica que se encuentra en un estado de espera para pasar al siguiente número y por lo tanto, la suma se mantiene igual.

La MTD realiza la misma operación para los siguientes dos símbolos "1" y "1" del segundo número, llevándonos a pasar por los estados $q_{3,4}$ y $q_{4,5}$. Como en el segundo número, el primer símbolo leído es "1" y se encuentra en la posición más significativa, se suma dos al acumulado de la suma, dándonos un valor de cuatro. Y en el último símbolo, igualmente, por encontrarse en la posición menos significativa, su valor es de uno y se suma a los cuatro que llevábamos, dándonos como resultado de la suma de los dos números binarios el valor de cinco.

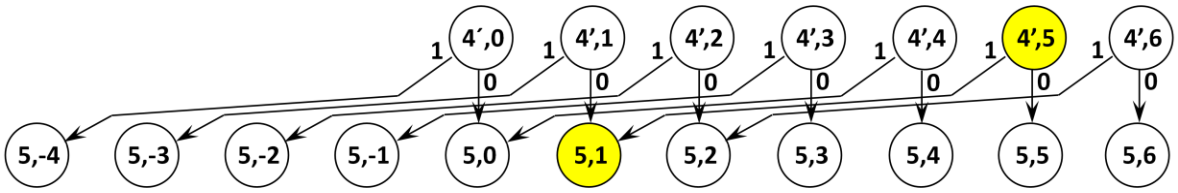
Después de que la MTD realiza la suma, pasa a la siguiente celda que contiene el símbolo de separación ",", lo lee y pasa al siguiente estado $q_{4',5}$, para después mandar la orden a la cabeza de moverse una celda a la derecha.

En la siguiente parte del programa, se trata de determinar si la suma obtenida de los dos números binarios es menor o igual al número binario de tres dígitos, y para esto realiza

una resta entre la suma y el número binario para determinar si es un ejemplar-sí o ejemplar-no.

Esta resta se realiza mediante la transición de estados debido a las limitaciones de la MTD, cada estado subsecuente tendrá en su subíndice el paso del programa en el que se encuentra y el valor actual de la resta.

En el ejemplar evaluado, después de evaluar la celda del último separador, la cabeza se mueve a la siguiente celda que contiene el símbolo “1” en la posición más significativa del número binario.




| | | | | |
|---|---|---|---|---|
| $q_{4',5}$ | $(q_{5,5}, 0, +1)$ | $(q_{5,1}, 1, +1)$ | $q_I$ | $q_I$ |

Figura A.6.- Función de transición que resta cuatro a la suma de cinco.

La función de transición nos lleva al estado $q_{5,1}$, (figura A.6) el cual indica que el programa está en el paso cinco con un valor de uno debido a que se le resta cuatro a la suma de los dos números binarios. Luego el CEF ordena a la cabeza que escriba el símbolo “1” en la celda actual y después se mueva una celda hacia la derecha.

El proceso se repite para los pasos 6 y 7 en donde, debido a la función de transición, el programa pasa por los estados $q_{6,-1}$ y $q_{7,-1}$, con un total en la resta de -1. Como terminamos con un valor negativo, significa que la suma de los dos números es menor al tercer número, por lo cual es un ejemplar-sí y termina en el estado $q_Y$.

# Anexo B. Máquina de Turing no determinista

En este anexo se muestra un ejemplo del diseño de una máquina de Turing no determinista (MTND) aplicado a un problema en particular con el fin de comprender mejor el funcionamiento de estas máquinas. El diseño de la MTND se realizó en base a la definición de la MTND dada en [Garey & Johnson, 1979].

## B.1 Ejemplo de una máquina de Turing no determinista

Para el entendimiento del siguiente ejemplo es necesario tener claros los conceptos de autómata finito no determinista y máquina de Turing no determinista (véase sección 2.5).

El problema abordado para diseñar la MTND fue el del circuito hamiltoniano (CH), el cual se describe a continuación:

El circuito hamiltoniano es un circuito que pasa por todos los vértices de un grafo una sola vez, con excepción del vértice inicial, en el cual comienza y termina el circuito.

**Ejemplar**: un grafo $G = (V, E)$.

**Pregunta:** ¿$G$ contiene un circuito hamiltoniano CH, es decir, un orden $<v_1, v_2, \ldots, v_n>$ de los vértices de $G$ donde $n = |V|$ y de modo que: $\{v_n, v_1\} \in E$ y $\{v_i, v_{i+1}\} \in E \ \forall i$, $1 \leq i < n$?

El objetivo de este ejemplo es diseñar una MTND para determinar si hay un circuito hamiltoniano en el grafo de la figura B.1:

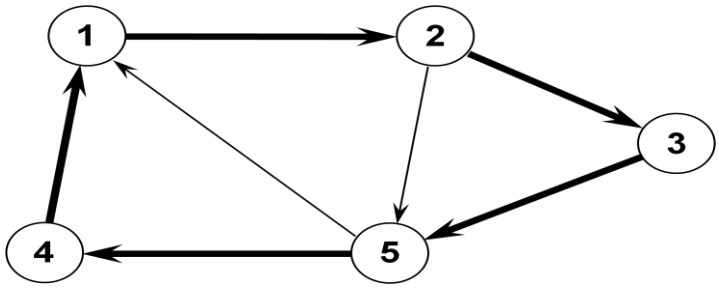



Figura B.1.- Grafo del problema a resolver por la MTD.

La matriz de adyacencia del grafo queda de la siguiente manera:

Tabla B.1.- Matriz de Adyacencia.

| | $v_1$ | $v_2$ | $v_3$ | $v_4$ | $v_5$ |
|---|---|---|---|---|---|
| $v_1$ | 0 | 1 | 0 | 0 | 0 |
| $v_2$ | 0 | 0 | 1 | 0 | 1 |
| $v_3$ | 0 | 0 | 0 | 0 | 1 |
| $v_4$ | 1 | 0 | 0 | 0 | 0 |
| $v_5$ | 1 | 0 | 0 | 1 | 0 |

Codificación del problema:

Tomando como base la matriz de adyacencia, el problema se puede codificar en la cinta de la siguiente manera:

| $b$ | 0 | 1 | 0 | 0 | 0 | $b$ | 0 | 0 | 1 | 0 | 1 | $b$ | 0 | 0 | 0 | 0 | 1 | $b$ | 1 | 0 | 0 | 0 | 0 | $b$ | 1 | 0 | 0 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | . | 29 | 30 |

Figura B.2.- Problema codificado en la cinta.

El problema empieza desde la celda uno de la cinta, en donde cada fila de la matriz de adyacencia está delimitada por una celda con un símbolo blanco.

### B.1.1 El módulo de adivinanza

El módulo de adivinanza selecciona aleatoriamente del conjunto {1, 2, …, 5} una secuencia de cinco dígitos correspondientes a los vértices del grafo. Esta secuencia constituirá la solución candidata del problema. El módulo emplea un autómata finito no determinista el cual se ilustra en la figura B.3.

El AFD emplea la notación $s_1 \rightarrow s_2$, $m$ para representar las acciones de lectura, escritura y movimiento.

1.- $s_1$ representa el símbolo que lee el módulo de adivinanza en la celda actual. El símbolo $\varepsilon$ en esta posición indica que la cabeza no efectúa lectura alguna; es decir, efectúa una transición espontáneamente.

2.- $s_2$ representa el símbolo que la cabeza escribirá en la celda actual. El símbolo $\varnothing$ indica que la cabeza no efectúa escritura alguna.

3.- $m$ representa el movimiento de la cabeza. Si $m$ = +1, se mueve una celda a la derecha, si $m$ = $-1$, se mueve una celda a la izquierda, si $m$ = 0 entonces no realiza movimiento, es decir, se mantiene en la misma celda.

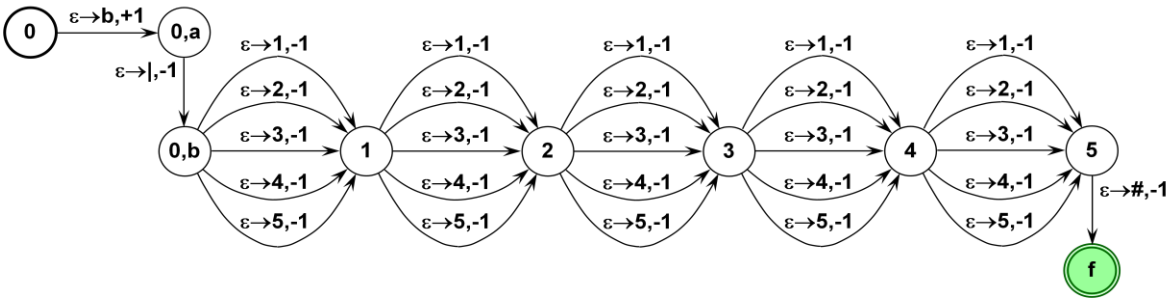



Figura B.3.- Diagrama del AFD del módulo adivinador.

La cabeza lectora del módulo de adivinanza se ubica en la posición cero de la cinta y escribe el símbolo "|" utilizado como delimitador para facilitar el cálculo del programa verificador.

En la figura B.3 se observa cómo la transición del estado $q_{0,a}$ al estado $q_{0,b}$ ordena a la cabeza no leer nada, escribir el símbolo "|" y moverse una celda a la izquierda. Esta transición ocurre cuando la MTND se encuentra en la celda cero de la cinta.

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| *b* | *b* | *b* | # | *4* | *5* | *3* | *2* | *1* | \| | *b* | 0 | 1 | 0 | 0 | 0 | *B* | 0 | 0 | 1 | 0 | 1 | . | . | . | 1 | 0 | 0 | 1 | 0 |
| | | | | -5 | | | | | 0 | | | | | 5 | | | | | 10 | | | | | | | | | | 30 |

Figura B.4.- Contenido de la cinta al finalizar el módulo de adivinanza.

Al ubicarse en la posición $-1$ de la cinta, la cabeza escribe aleatoriamente un símbolo que corresponde a un vértice del grafo. Esto está representado en el AFD en la transición del estado $q_{0,b}$ al estado $q_1$ mediante cinco posibles transiciones, de las cuales en ninguna ordena leer un símbolo en la celda $-1$, y solamente escribe. Cada una de las posibles

transiciones escribe un símbolo distinto que represente un vértice del grafo, dando así la cualidad de aleatoriedad al AFD. Por último se le ordena a la cabeza moverse una celda a la izquierda. Esto se repite durante la transición del estado $q_1$ al estado $q_5$ escribiendo cinco símbolos que representan cinco vértices en las celdas que van desde la $-1$ a la $-5$. La ultima transición entre el estado $q_5$ al estado $q_f$ indica la finalización del proceso de adivinación de una solución candidata, y escribe el símbolo "#" en la celda $-6$ para delimitar la solución y facilitar el proceso de verificación.

### B.1.2 El módulo de verificación

El módulo de verificación tiene como objetivo las siguientes tareas:

1. Comprobar si el ejemplar es válido.
2. Comprobar que la secuencia de vértices generados no tenga elementos repetidos.
3. Comprobar que para cada par de vértices consecutivos ($v_i$, $v_{i+1}$) de la secuencia se cumple que ($v_i$, $v_{i+1}$) sea una arista de $G$ y que ($v_1$, $v_5$) sea también una arista de $G$.

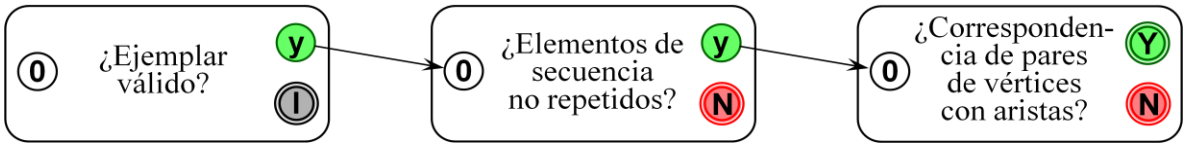



Figura B.5.- Representación por submódulos de las tareas del módulo de verificación.

#### B.1.2.1 ¿Ejemplar valido?

El siguiente pseudocódigo tiene como objetivo validar el ejemplar.

Definir $i$ = cinta($-7$)

**1.** Hacer $i = 0$.
**2.** Si la fila $i$ válida, pasar al paso 3, de lo contrario parar en el estado $q_I$.
**3.** Hacer $i = i + 1$.
**4.** Si $i \leq 5$ ir al paso 2; si $i = 6$ ir al estado "$y$".

Se añade la variable $i$ que tendrá la función de contador, el cual aumentará cada vez que se revise la validez de una fila de la matriz de adyacencia. El valor de $i$ será almacenado en la celda $-7$ de la cinta.

**Validación de ejemplar: Paso 1**

Como durante el proceso de revisión de las filas será necesario incrementar la variable $i$, se creó un submódulo que posicionará la cabeza de la MTND en la celda $-7$ para poder actualizarla. La figura B.6 muestra el submódulo antes mencionado.

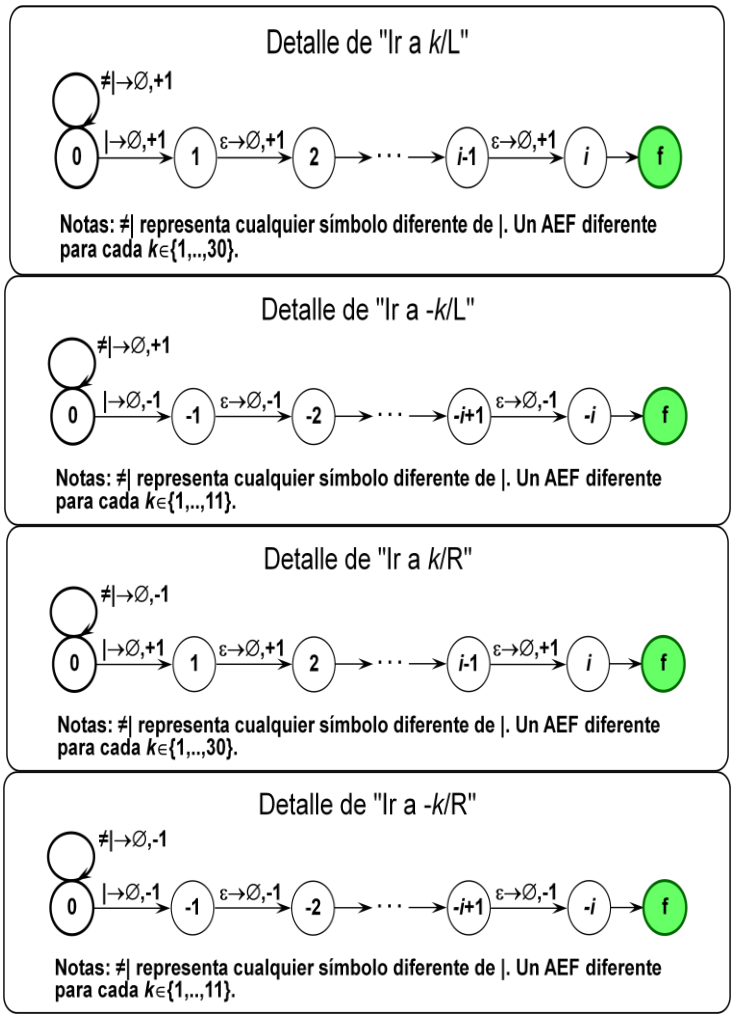



Figura B.6.- Diagrama general de los submódulos que posicionan la cabeza en cualquier celda de la cinta.

En la figura B.6 se describen cuatro submódulos que posicionan la cabeza en cualquier celda de la cinta. La primera figura supone que la cabeza se encuentra en una celda menor a cero, por lo que primero mueve la cabeza hacia la derecha mientras no encuentre el símbolo "|" que se encuentra en la celda cero, y después de llegar a ésta, mueve la cabeza hacia la izquierda (no lee ni escribe símbolos) hasta que llega a la celda deseada, que en nuestro caso sería la celda $-7$. La segunda imagen realiza la misma función, sólo que una

vez encontrado el símbolo "|", mueve la cabeza hacia la derecha para encontrar una celda específica *k* positiva. La tercera imagen corresponde al submódulo que posiciona la cabeza en una celda positiva partiendo de otra celda positiva por lo que a diferencia de los dos módulos anteriores, mueve la cabeza hacia la izquierda para encontrar la celda cero con el símbolo "|" y de ésta mueve la cabeza hasta la celda $k$ positiva. La última imagen posiciona la cabeza en una celda positiva partiendo de una celda negativa, por lo que a diferencia de la primera, la cabeza se moverá hacia la derecha para encontrar la celda cero con el símbolo "|" y después hacia la derecha para encontrar la celda $k$ positiva.

**Validación de ejemplar: Paso 2**

Primero, la cabeza se mueve hacia la celda $-7$ para leer el valor de la variable $i$. Después, la cabeza se mueve al inicio de la fila $i$ (de la matriz de adyacencia) para empezar a validar la fila.

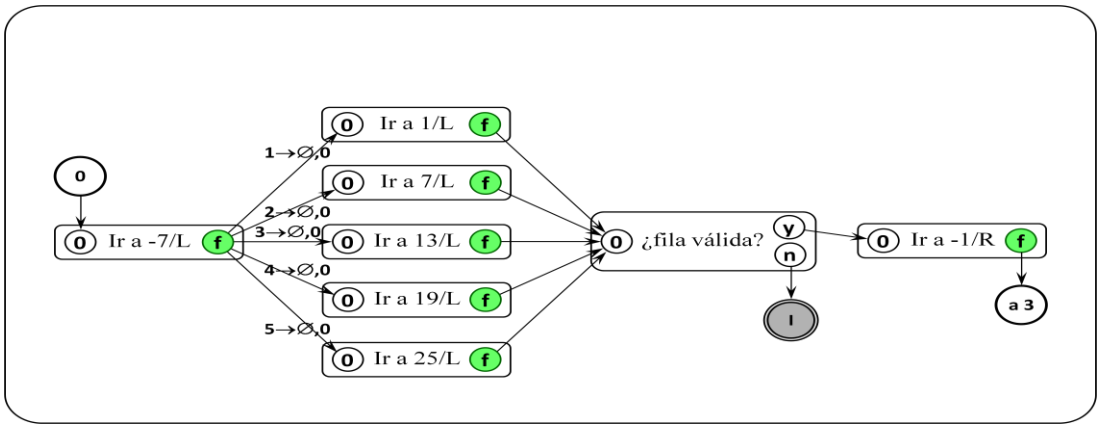



Figura B.7.- Módulo general de validación de fila. Primero determina qué fila evaluará y después se posiciona en la fila a evaluar.

La figura B.7 muestra la trayectoria que sigue la cabeza para posicionarse en el inicio de la fila de la matriz de adyacencia a validar. Primero se ubica en la celda $-7$ para leer el valor de $i$. Después la cabeza se ubica en el inicio de la fila $i$. Enseguida pasa al módulo de verificación de fila y determina si la fila es válida o no. Si la fila es inválida, llega a un estado $q_I$ y termina el programa y en caso de ser válido, posiciona la cabeza de nueva cuenta en la celda $-7$ para actualizar el valor de $i$ y pasar a la siguiente fila.

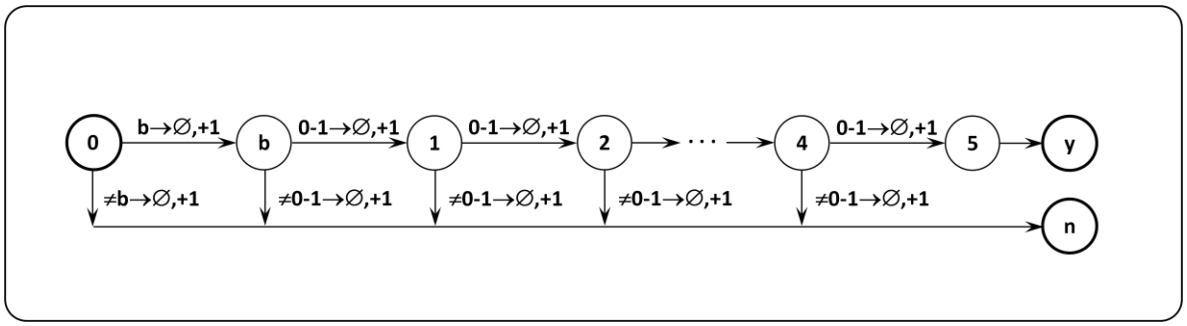



Figura B.8.- Módulo de verificación de fila.

En la figura B.8 se muestra el módulo de verificación de la fila $i$, en donde en la primera celda, cualquier símbolo leído que sea diferente de "$b$" hace inválida la fila y pasa al estado de negación $q_n$, en caso contrario continúa y la cabeza se mueve a la siguiente celda. Si en las siguientes cinco celdas, se lee un carácter diferente a "0" o "1" pasa al estado de negación $q_n$ y termina el programa, en caso contrario continúa y pasa al estado de aceptación local $q_y$.

**Validación de ejemplar: Paso 3**

Para aumentar el valor de $i$ primero se posiciona la cabeza en la celda $-7$, y se efectúa una de cinco posibles transiciones para cambiar el valor de la variable $i$. La figura B.9 muestra el módulo de incremento de la variable $i$.

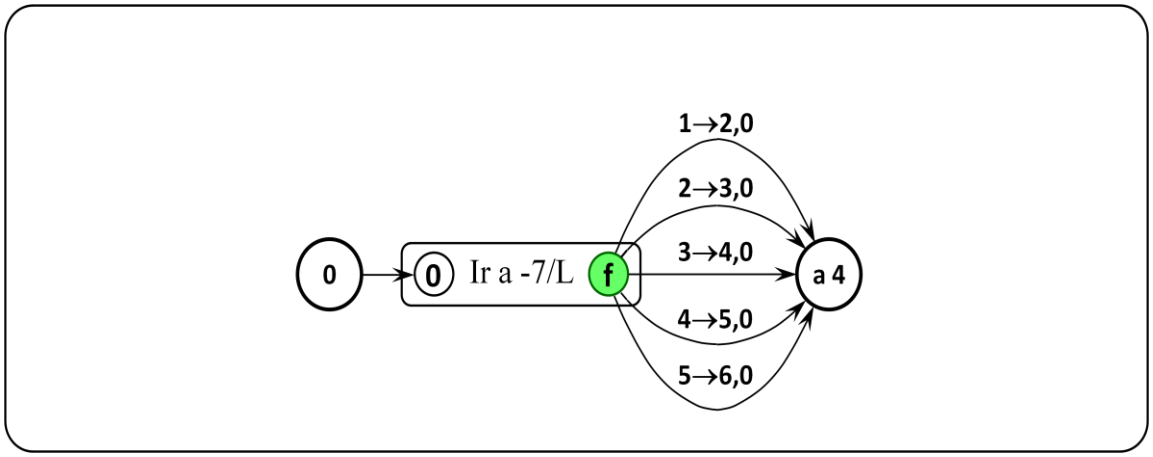



Figura B.9.- Módulo de incremento de la variable *i*.

En la figura B.9, la función de transición primero lee el valor de la celda $-7$, después escribe en la celda el valor leído más 1. Por ejemplo, si lee un "1", escribe un "2".

**Validación de ejemplar: Paso 4**

En el paso cuatro primero es necesario mover la cabeza a la celda $-7$, si $i \leq 5$ pasa de nuevo al paso dos, de lo contrario termina en el estado de aceptación local $q_f$ (figura B.10).

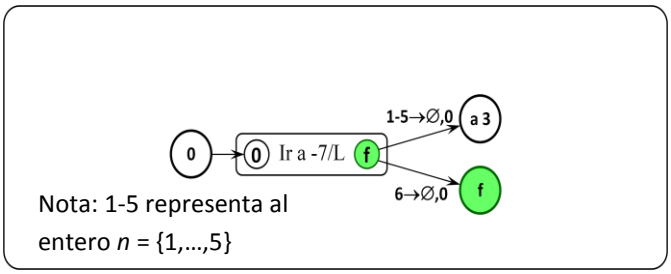



Figura B.10.- Módulo para determinar si continúa o termina la validación del ejemplar.

### B.1.2.2 ¿Hay vértices repetidos?

El siguiente es el pseudocódigo del módulo de elementos no repetidos.

Definir variables $i$ = cinta($-7$), $j$ = cinta($-8$), $v_i$ = cinta($-9$), $v_j$ = cinta($-10$), $oc$ = cinta($-11$). (Nota: $oc$ representa el número de ocurrencias de un vértice en la secuencia aleatoria de vértices.)

1.- Hacer $i = 1$.

2.- Hacer $oc = 0$.

3.- Hacer $j = 1$.

4.- Copiar valor de cinta($-i$) a $v_i$.

5.- Copiar valor de cinta($-j$) a $v_j$.

6.- Si $v_i = v_j$, hacer $oc = oc + 1$ y si $oc$ adquiere el valor 2, parar en el estado "$N$".

7.- Hacer $j = j + 1$.

8.- Si $j \leq 5$ ir al paso 5; si $j = 6$ hacer $i = i + 1$.

9.- Si $i \leq 5$ ir al paso 3; si $i = 6$ ir al estado "$y$".

**Elementos no repetidos: Pasos 1, 2, 3**

En la figura B.11 podemos observar los submódulos que inicializan las variables $i$, $j$, $oc$ en uno. Simplemente cada módulo mueve la cabeza lectora hacia la celda especificada, $-7$ para $i$, $-8$ para $j$, y $-11$ para $oc$.

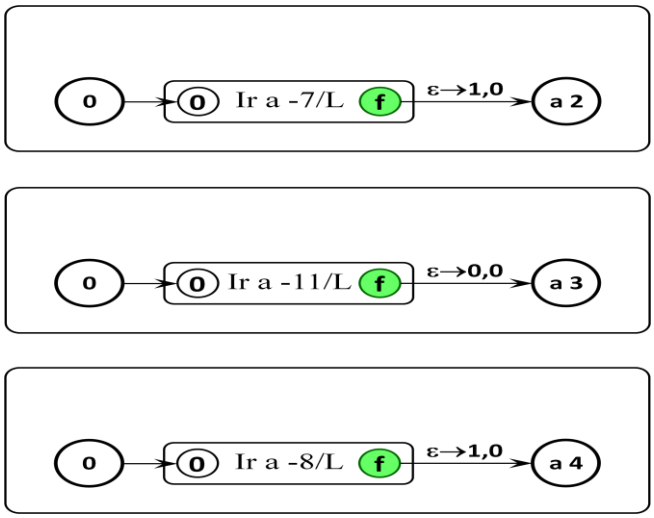



Figura B.11.- Módulos de iniciación de las variables *i*, *j*, *oc*.

**Elementos no repetidos: Pasos 4 y 5**

Después de inicializar las variables, copiamos el valor almacenado en las celdas $-i$ y $-j$ a las variables $v_i$, $v_j$ que corresponden a las celdas $-9$ y $-10$ de la cinta. La figura B.12 muestra cómo primero se ubica la cabeza en la celda de la variable $i(-7)$ y después se dirige a la celda $-i$ para copiar el valor en la celda correspondiente a la variable $v_i$ $(-9)$.

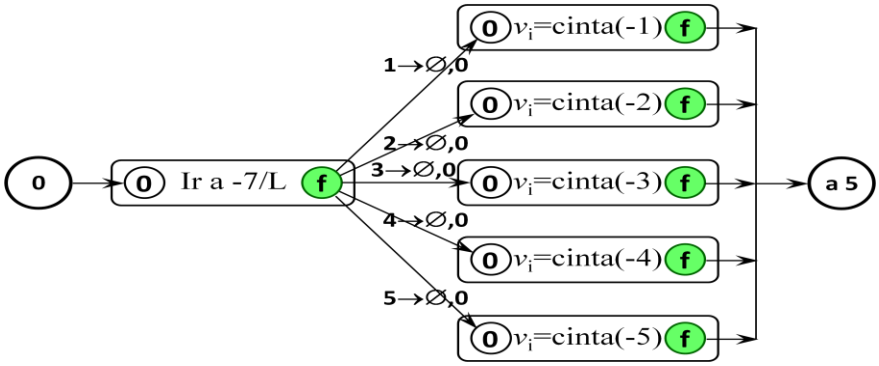



Figura B.12 Módulo que copia el valor de la celda $-i$ en la variable $v_i$.

El módulo $v_i$=cinta($-i$) realiza la función de leer el valor de la celda $-i$ y mover la cabeza a la posición $-9$ para escribir el símbolo leído en la celda $-i$. La figura B.13 muestra este submódulo.

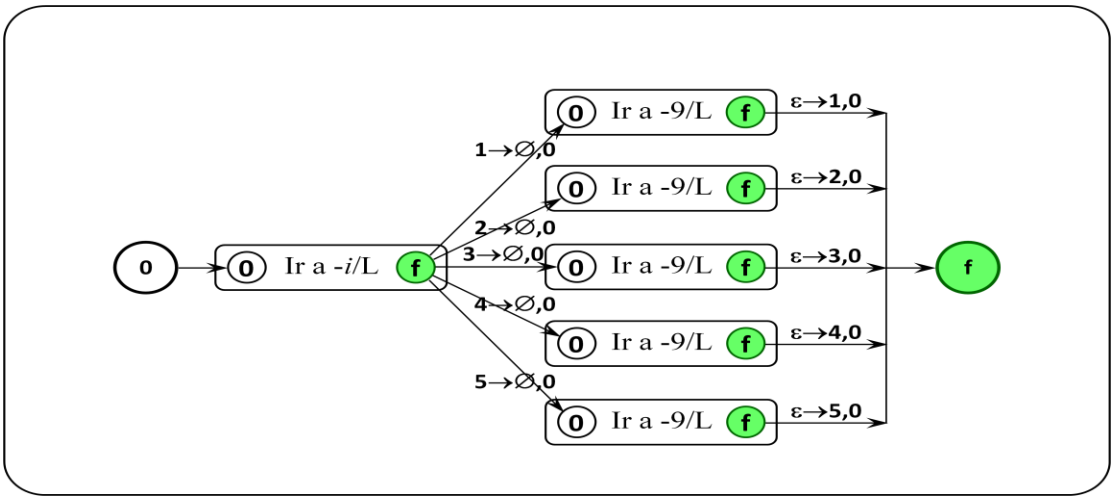



Figura B.13.- Submódulo cinta(–*i*) que lee el valor de la celda –*i* y lo copia en la variable $v_i$ (–9).

Para copiar el valor de la celda –*j* en la variable $v_j$ (paso 5) es básicamente el mismo procedimiento, sólo cambian las posiciones de las celdas de las variables, en este caso sería –8 para la variable *j*, y –10 para la variable $v_j$.

**Elementos no repetidos: Paso 6**

El paso seis está descrito de manera general en la figura B.14, en donde se observa un módulo compuesto por varios submódulos. Primero hay un submódulo que tiene como función la de comprobar si $v_i = v_j$ y tiene dos estados de finalización, un estado sí y un estado no. Si termina en el estado sí (*y*), aumenta en uno el valor de la variable *oc* y si es igual a uno, continúa el proceso hacia el paso 7, pero si es igual a dos, para en un estado de negación "$q_N$". Si $v_i$ y $v_j$ son diferentes, pasa directamente al paso 7.

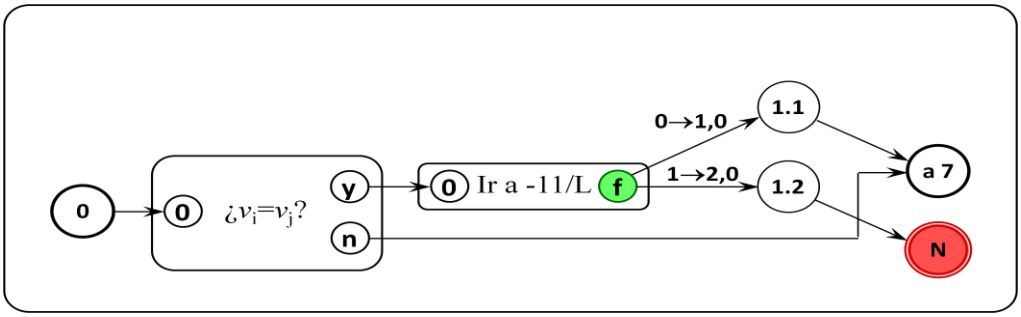



Figura B.14.- Diagrama del módulo del paso seis del pseudocódigo de vértices repetidos.

Para comprobar si las variables $v_i$ y $v_j$ son diferentes, primero copia el valor de la variable $v_i$ en la celda –9 para luego mover la cabeza hacia la celda –10 en donde se almacena el valor de la variable $v_j$ para verificar si es igual al valor de la variable $v_i$. De ser iguales el

módulo llega al estado "$q_y$" que indica que sí son iguales, de lo contrario llega al estado "$q_n$". Este submódulo se describe en la figura B.15.

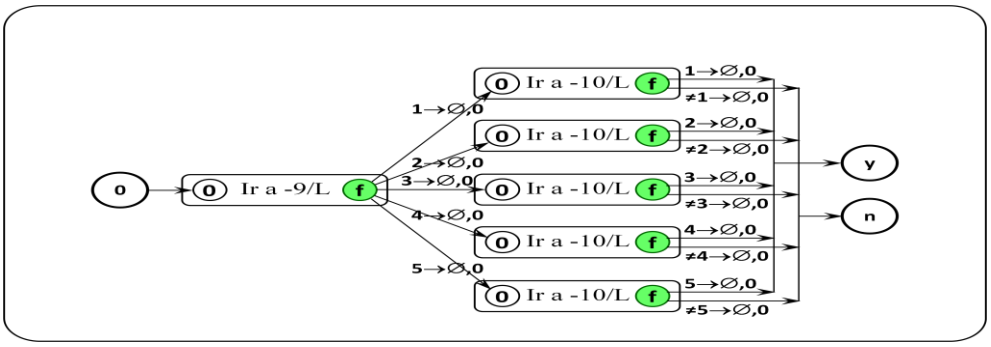



Figura B.15.- Submódulo ¿$v_i = v_j$?

**Elementos no repetidos: Paso 7**

Para realizar el paso 7 solamente se ubica la cabeza en la celda de la variable $j$ (–8) y dependiendo del valor leído, se escribe el valor leído más 1 ($j$ + 1). La figura B.16 ilustra este proceso.

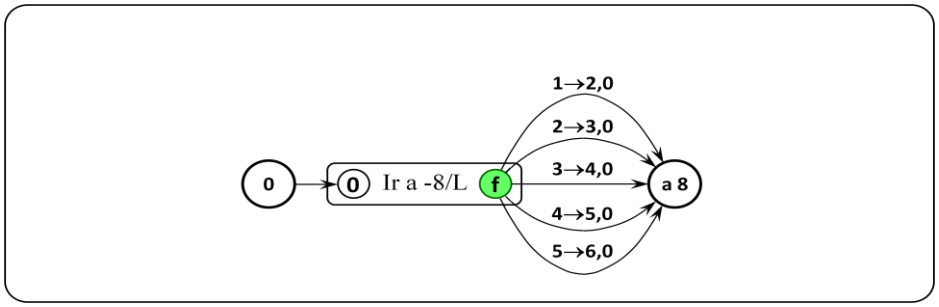



Figura B.16.- Módulo $j = j + 1$.

**Elementos no repetidos: Paso 8**

Para el paso ocho es necesario verificar si $j \leq 5$ y para esto se desarrolló un módulo en el cual, primero se ubica en la celda correspondiente a la variable $j$ (–8) y después la cabeza lee el valor almacenado. Si el valor almacenado está en el rango, regresa al paso 5, de lo contrario, la cabeza se mueve hacia la celda de la variable $i$ (–7) y aumenta su valor a $i$ + 1.

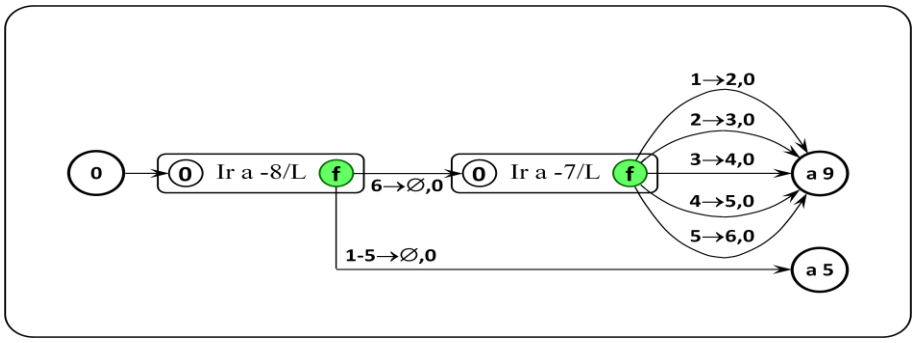



Figura B.17.- Diagrama del módulo del paso 8.

**Elementos no repetidos: Paso 9**

El noveno y último paso verifica si $i \leq 5$, y para esto se crea un módulo en el cual primero se ubica la cabeza en la celda de la variable $i$ (–7), y si el valor leído esta dentro del rango, regresa al paso tres, y en caso contrario, el programa termina en el estado de aceptación "$q_f$" y se concluye que no hay vértices repetidos en la secuencia aleatoria.

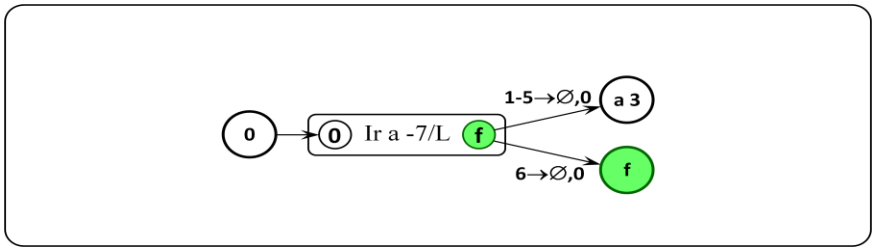



Figura B.18.- Módulo representativo del paso 9.

Con esto termina el módulo de verificación de vértices repetidos para dar paso al módulo siguiente: el módulo de verificación de correspondencia de aristas.

### B.1.2.3 ¿Existe una artista entre dos vértices consecutivos?

Pseudocódigo del módulo de correspondencia de aristas.

Definir $i$ = cinta(–8), $j = i + 1$, $fila_1 = 1$, $fila_2 = 7$, $fila_3 = 13$, $fila_4 = 19$, $fila_5 = 26$

1.- Copiar valor de cinta(–1) en cinta (–6).

2.- Hacer cinta(–7) = #.

3.- Hacer $i = 1$.

4.- Copiar valor de cinta(–*j*) a cinta($fila_1$), cinta($fila_2$), cinta($fila_3$), cinta($fila_4$), cinta($fila_5$).

5.- Ir a la posición $fila_i$ e ir a la cinta($fila_i$) posiciones a la derecha; si el contenido es "1", ir al paso seis, de lo contrario, parar en el estado "$N$".

6.- Hacer $i = i + 1$.

7.- Si $i \leq 5$ ir al paso 4; si $i = 6$ parar en el estado "$Y$".

**¿Existe Arista?: Paso 1**

En el paso uno se copia el primer vértice de la secuencia aleatoria de vértices (es decir, el valor de la celda (–1)) al final de la secuencia (es decir, a la celda (–6)), y para esto se mueve la cabeza a la celda –1, para leer el valor e imprimirlo en la celda (–6). Este paso está descrito por el AFD de la figura B.19.

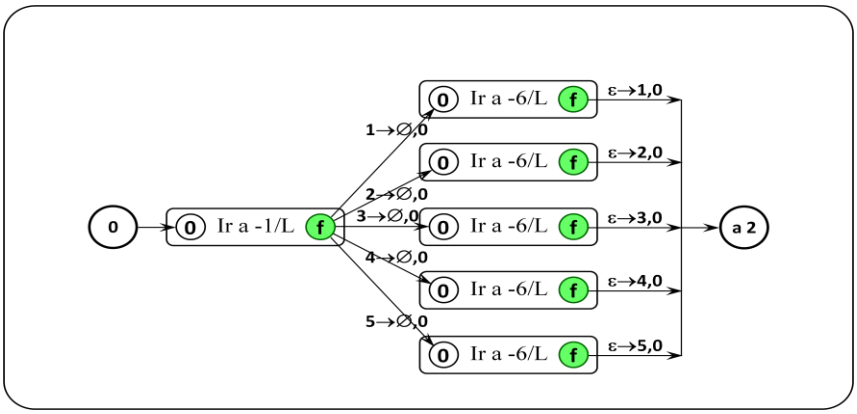



Figura B.19.- Módulo que copia el valor de la celda (–1) a la celda (–6).

**¿Existe Arista?: Pasos 2 y 3**

En el paso dos se imprime el símbolo "#" en la celda (-7) y en el paso tres se inicializa la variable $i$ al escribir el símbolo "1" en la celda (-8).

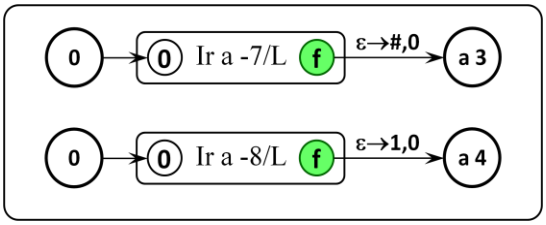



Figura B.20.- Módulos descriptivos del pasos 2 y del paso 3.

**¿Existe Arista?: Paso 4**

En este paso se realiza el copiado del valor de la celda (–*j*) a las celdas (*fila*$_1$), (*fila*$_2$), (*fila*$_3$), (*fila*$_4$) y (*fila*$_5$). Para tal efecto, primero mueve la cabeza a la posición de la variable *j* (–8) y lee el valor y se mueve a la celda (–*j*) para copiar su valor en las celdas (1), (7), (13), (19) y (26) correspondientes al valor de las variables *fila*$_i$.

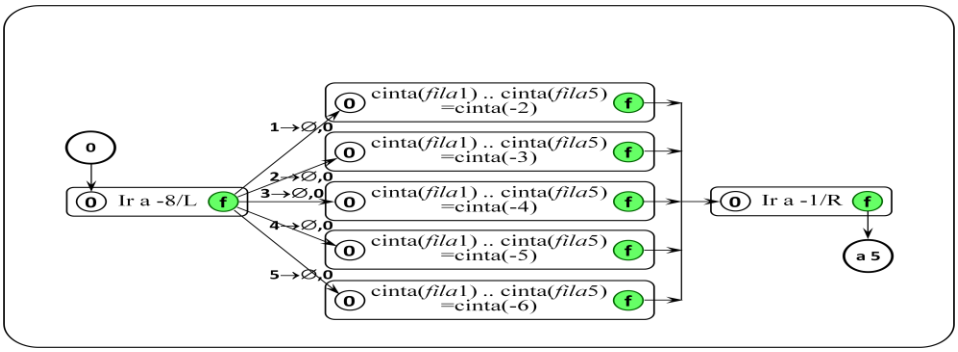



Figura B.21.- Descripción general del módulo correspondiente al paso 4.

En la figura B.21, los submódulos de cinta(*fila*) tienen la función de moverse hacia la celda (–*j*) para leer su símbolo y copiarlo en las celdas (1), (7), (13), (19) y (26). La figura B.22 muestra la representación de este submódulo.

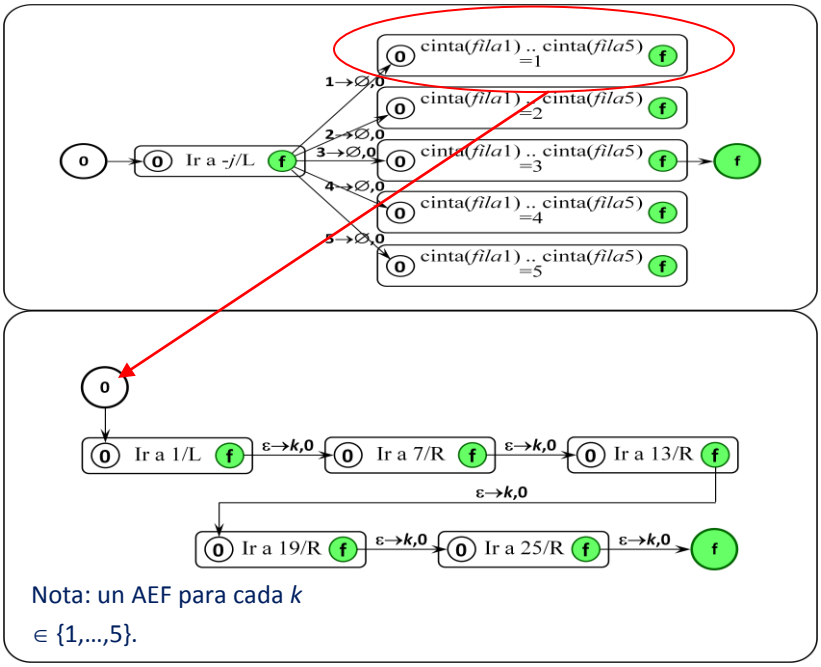



Figura B.22.- Submódulo del paso 4.

**¿Existe Arista?: Paso 5**

En este paso, la cabeza se posiciona en la celda (*fila*$_i$) lee el valor y se mueve un número de celdas a la derecha determinadas por el valor leído. Entonces verifica si el valor encontrado en tal celda es "1" para poder pasar al paso seis, en caso contrario llega a un estado "$q_N$ ".

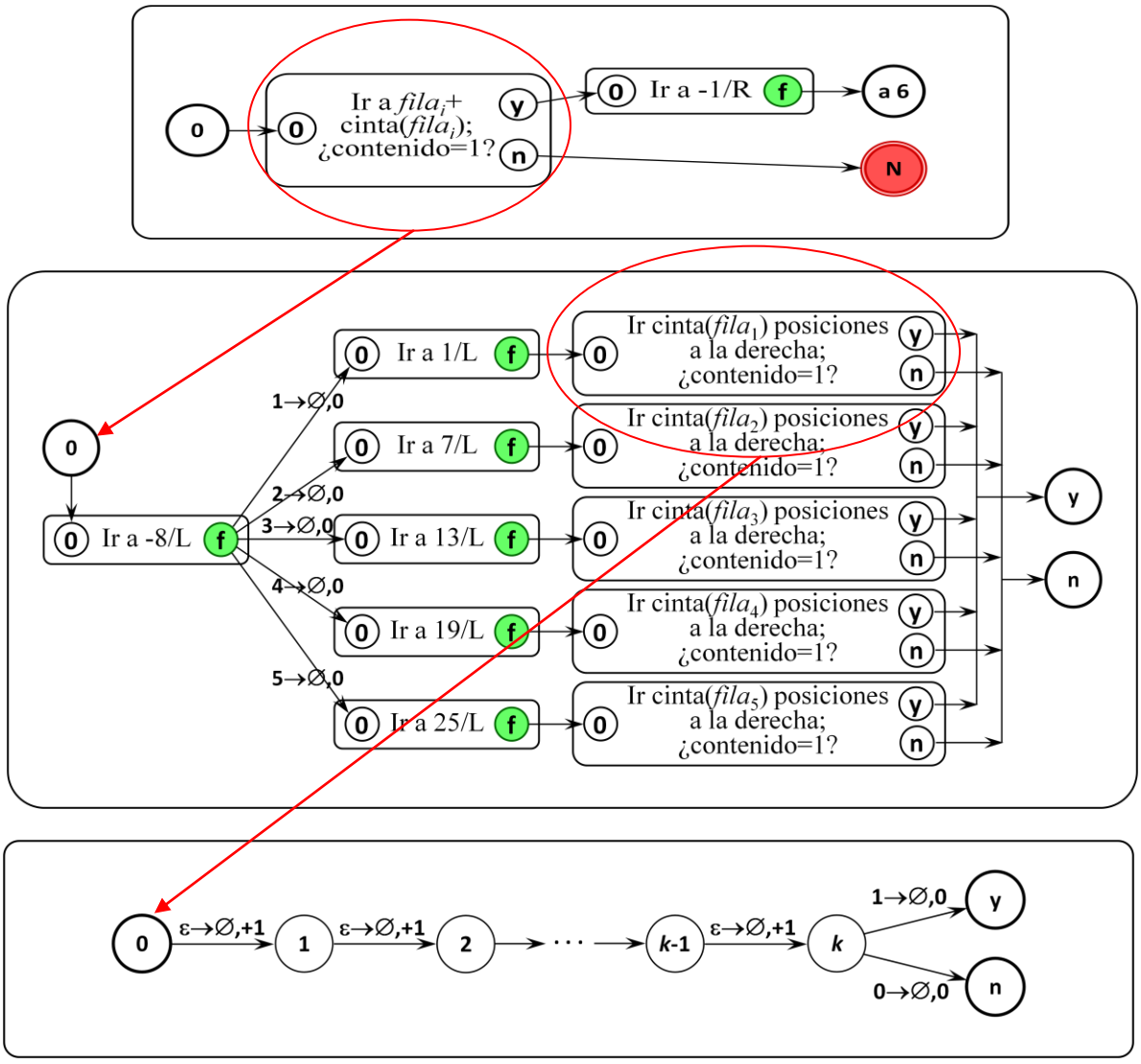



Figura B.23.- Diagrama del módulo que se encarga de determinar si el arco entre dos vértices contiguos existe.

En la figura B.23 se puede observar el módulo completo, en el que primero lee el valor de la variable $i$ (–8) para determinar el subíndice de la variable *fila*$_i$. Después se dirige la cabeza hacia la celda (*fila*$_i$), que es el inicio de la fila $i$ de la matriz de adyacencia. Después lee el valor de la celda (*fila*$_i$) para determinar cuántas celdas a la derecha se moverá la

cabeza. Una vez posicionada la cabeza en la celda correspondiente a ($v_i$, $v_{i+1}$) donde $v_i$ y $v_{i+1}$, corresponden al par de vértices consecutivos examinados, lee el valor de la celda. Si lee un símbolo "1", entonces ($v_i$, $v_{i+1}$) realmente es una arista del grafo y pasa a un estado $q_y$ y continúa hacia el paso seis, de lo contrario pasa al estado $q_N$ y termina el programa determinando que es un ejemplar-no.

**¿Existe Arista?: Paso 6**

El paso seis se encarga de incrementar la variable $i$ una unidad. La figura B.24 muestra el módulo de este paso.

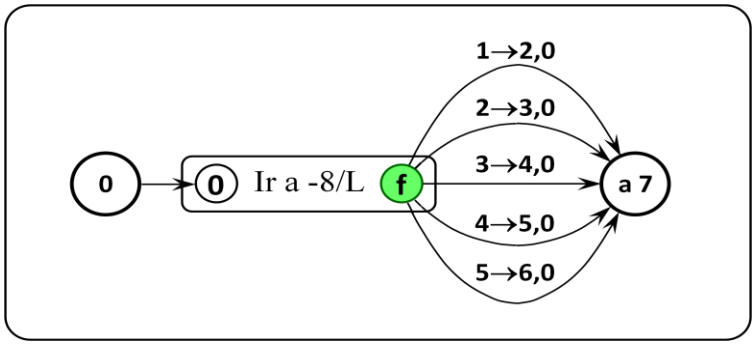



Figura B.24.- Módulo que incrementa la variable *i*.

**¿Existe Arista?: Paso 7**

El séptimo y último paso de este módulo verifica si la variable $i$ no es mayor a cinco. Si no es mayor a cinco, regresa al paso cuatro del módulo, de lo contrario, pasa al estado de aceptación "$q_Y$" y termina el proceso aceptando el ejemplar.

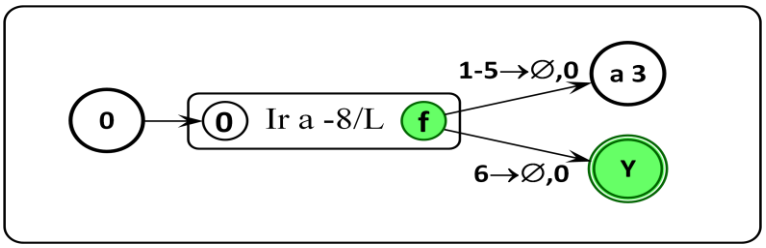



Figura B.25.- Módulo del paso final que verifica que todas las aristas de la solución hayan sido revisadas.

Una vez finalizado este módulo, la MTND termina su proceso aceptando el ejemplar como un ejemplar-sí.

# Bibliografía

[S.A. Cook, 1971] S.A. Cook, **"**The complexity of theorem-proving procedures", *In STOC '71: Proceedings 3rd Annual ACM symposium on theory of computing*, Association for Computing Machinery, pp. 151-158. New York 1971 doi: doi.acm.org/10.1145/800157.805047.

[T.H. Cormen, 1990] T.H. Cormen, Leiserson C.E., Rivest R.L., Stein C. *Introduction to algorithms*, 2da. edición MITPress, McGraw-Hill, USA, 2001.

[Garey & Johnson, 1979] M.R. Garey, D.S. Johnson, *Computers and intractability: A guide to the theory of NP-completeness*, W.H. Freeman and Company, New York, 1979.

[O. Goldreich, 2008] O. Goldreich, *P, NP, and NP-Completeness: The Basics of Computational Complexity*, Depatament of Commputer Science and Applied Mathematics Weizmann Institute of Science , Rehovot, ISRAEL, 14/06/2008.

[R. Greenlaw, 1995] R. Greenlaw, *Limits to parallel computation: P-Completeness theory*, 1era. edición, Oxford University Press, New York, 06/04/ 1995.

[R. Karp, 1972] R.M. Karp, "Reducibility among combinatorial problems", *Complexity of Computer Computations*, R.E. Miller and J.W. Thatcher eds. Plenum Press, pp. 85-103, New York, 1972.

[A. Maheshwari, 2010] A. Maheshwari, M. Smid, *Introduction to Theory of Computation*, Lecture Notes COMP 2805, School of Computer Science Carleton University, 01/04/2010.

[D. M. Mount, 2003] D. M. Mount, *Design and Analysis of Computer Algorithms*, Lecture Notes CMSC 451, Department of Computer Science University of Maryland, 2003.

[A.L. Rosenberg, 2010] A. L. Rosenberg, *The pillars of computation theory*, 1era. edición, Springer, New York, 11/11/ 2009.

[J.A. Ruiz Vanoye, 2008] J. A. Ruiz-Vanoye, *Desarrollo de Indicadores de Casos Aplicables a la Selección de Algoritmos en el problema 2-Partition*, Tesis Doctoral, Departamento de Ciencias Computacionales, Centro Nacional de Investigación y Desarrollo Tecnológico, 26/10/2008.

[E. Schaeffer, 2007] E. Schaeffer, *Complejidad computacional de problemas y el análisis y diseño de algoritmos*, MITPress, Mexico, 2007.

[M. Sipser, 2006] M. Sipser, *Introduction to the theory of computation*, Massachusetts Institute of Technology, Thomson Course Technology, Boston. Massachusetts, 2006.

[Tilburg University] Tilburg University. "*Polynomial reduction and NP-completeness"*.Class notes from: http://www.tilburguniversity.edu/research/institutes-and-research-groups/center/staff/haemers/reader09ico.pdf

[L. Trevisan, 1997] L. Trevisan, *Reductions and (Non-) Approximability*, Tesis Doctoral, Universidad de Roma "La Sapienza", Roma, Italia, 07/09/ 1997.