Universidad Autónoma Metropolitana

Posgrado en Ciencias y Tecnologías de la Información

Sobre la computabilidad del clan-comportamiento

Tesis Doctoral

María del Carmen Cedillo Chagoya

Director:
Miguel Ángel Pizaña López

14 de abril de 2021

Dedicada a mi madre:

por impulsarme a
ser mejor cada día.

Agradecimientos

Quiero agradecer a todos los que estuvieron conmigo en el proceso de la investigación reflejada en este documento. Cada uno de ustedes aportó algo bueno, ya sea en conocimientos, en experiencias, en sabiduría y en confianza a lo que estaba realizando.

Gracias a mi familia, especialmente a mi mamá y a mi hermana, que siempre me dieron la fortaleza para concluir esto, ya sea con una palabra de aliento o con un gran abrazo. Gracias a mi pequeño sobrino Dylan por sus ocurrencias que hacian amenas las horas de trabajo y además por su curiosidad en las animaciones que se obtuvieron con las gráficas de clanes.

Agradezco la paciencia que mis amigos cercanos me dieron cuando intentaba explicarles los resultados de mi investigación, ya que quizá no me entendían pero me escuchaban. También, gracias a mis colegas y amigos de otras áreas que siempre estuvieron dispuestos a apoyarme para que me diera a conocer en otras instituciones académicas.

Gracias a mis profesores que permitieron intercambiar ideas sobre mi investigación. Muchas de ellas dieron como resultado la integración de cosas nuevas y el análisis profundo de otras. Gracias a mi tutor, Miguel Pizaña, por haber fomentado un ambiente adecuado para culminar con éxito esta investigación; además por la paciencia y comprensión brindada, y sin dudarlo, por ayudarme a obtener el perfil con el cual egreso del Doctorado. En verdad muchas gracias por confiar en mí y en mis capacidades.

Por último agradezco al CONACYT por haberme apoyado económicamente en mi instancia durante la mayor parte de estos estudios, y así mismo, a la Universidad Autónoma Metropolitana por los apoyos en las inscripciones y derivados.

Índice general

Agradecimientos **III**

Índice de tablas **IX**

Índice de figuras **XV**

1. Introducción **1**

2. Antecedentes **5**
2.1. Teoría de gráficas 5
2.2. Gráficas de clanes 9
2.3. Teoría de autómatas 11
2.4. Lógica de primer orden 14
2.5. Estructuras automáticas 18
2.6. Máquinas de Turing 24
2.7. Teoría de la computabilidad 30

3. Configuraciones con estampa de tiempo **33**
3.1. Definición 33
3.2. Relación de transición en configuraciones con estampa de tiempo . . . 34

4. Indecibilidad del K-comportamiento para gráficas automáticas **41**
4.1. La indecibilidad de la clan-convergencia 41
4.2. Consecuencias de la indecibilidad de la clan-convergencia 52

5. Inexpresibilidad del K-comportamiento en lenguaje de primer orden **55**
5.1. Juegos de Ehrenfeucht-Fraïssé de k-rondas 55
5.2. Clanes y juegos de Ehrenfeucht-Fraïssé 56

6. Simulación de compuertas lógicas con gráficas de clanes **61**
6.1. Idea básica para el diseño de una computadora digital 61
6.2. Primera codificación 62

6.3. Segunda codificación 78

7. Simulación de las máquinas de Turing 81
7.1. Flip-flops 82
7.2. Unidad de control 85
7.3. Celda cero 89
7.4. i-ésima celda 91
7.5. Simulación de la ejecución de la máquina de Turing 93
7.5.1. Lectura 93
7.5.2. Cambio de estado 96
7.5.3. Escritura 99
7.5.4. Movimiento de cabeza 100
7.6. Ejemplo de simulación de la ejecución de la máquina de Turing 103
7.6.1. Lectura 104
7.6.2. Cambio de estado 107
7.6.3. Escritura 110
7.6.4. Movimiento de cabeza 112

8. Conclusiones 117

A. Código para la construcción de los componentes 119
A.1. Código de construcción 119
A.2. Código de exploración 130
A.3. Compilación 131
A.4. Código para la construcción de nuevos componentes 132

Bibliografía 139

Índice analítico 139

Índice de tablas

2.1. Las cadenas de los renglones 1 y 2 corresponden respectivamente a las configuraciones abp_1abb y $abap_0bb$. Ambas configuraciones pertenecen a la transición $abp_1abb \vdash abap_0bb$, donde M_P se mueve a la derecha. Sólo los símbolos son distintos en las columnas de las posiciones 3 y 4. 27

2.2. Las cadenas de los renglones 1 y 2 corresponden respectivamente a las configuraciones $ababbp_0$ y $ababp_2b_\sqcup$. Ambas configuraciones pertenecen a la transición $ababbp_0 \vdash ababp_2b_\sqcup$, donde M_P se mueve a la izquierda. Sólo los símbolos son distintos en las columnas de las posiciones $5, 6$ y 7. 27

3.1. Caso en donde $|t| = |t+1|$ en una transición de configuraciones con estampa de tiempo. La misma longitud en t permite que las columnas (correspondientes a cada posición) tengan símbolos pertenecientes a cadenas del mismo lenguaje, ya sea en $\mathtt{N}$ o en $\Gamma^*Q\Gamma^*$. 34

3.2. Caso en donde $|t| < |t+1|$ en una transición de configuraciones con estampa de tiempo. La distinta longitud en t provoca un ligero desfasamiento de símbolos en las columnas (correspondientes a cada posición), de tal manera que en la posición 4 $u_1 \in \Gamma$ y $0 \in \{0,1\}$. 35

3.3. Descripción de los movimientos en $\delta_3 \in M_3$ definidos en el lema 3.2.1 38

6.1. Definición de la compuerta `OR`. 64

6.2. Definición de la compuerta `AND`. 72

7.1. Definición del *flip-flop* SR. El símbolo "?" indica estado indefinido. . . 83

7.2. Definición del *flip-flop* D. 84

7.3. Función de transición δ de una máquina de Turing con el conjunto de estados $\{\mathtt{q}_0, \mathtt{q}_1, \mathtt{q}_2\}$, donde $\mathtt{q}_0$ es el estado inicial y $\mathtt{q}_2$ es el estado final, con alfabetos $\Sigma = \{0,1\}$ y $\Gamma = \{0,1\}$. 85

7.4. Configuración inicial de la unidad de control en el proceso de lectura para cuando la ejecución de la máquina de Turing está por comenzar. El valor de los *flip-flops* del conjunto estado nuevo no es importante. 94

7.5. Configuración inicial de la unidad de control en el proceso de lectura para cuando la ejecución de la máquina de Turing ya había comenzado, es decir, cuando el proceso anterior fue el movimiento de cabeza. El valor de los *flip-flops* del conjunto estado nuevo no es importante. . . 94
7.6. Configuración inicial de la celda cero y de cualquier i-ésima celda en el proceso de lectura. El asterisco (*) en H hace referencia que puede estar un 1 o un 0 según si la cabeza está o no en la celda, respectivamente. Mientras que d (dato almacenado en la celda) puede ser 0 o 1. 95
7.7. Configuración inicial de la unidad de control en el procesamiento de información. El valor del dato leído en la cinta es representado por d, el cual puede ser 0 o 1. 97
7.8. Configuración inicial de la unidad de control en el proceso de escritura. El valor a escribir en la cinta es representado por d', el cual puede ser 0 o 1. 99
7.9. Configuración inicial de la celda cero y de cualquier i-ésima celda en el proceso de escritura. El asterisco (*) en H hace referencia que puede estar un 1 o un 0 según si la cabeza está o no en la celda, respectivamente. Mientras que d es el valor del dato almacenado en la celda, el cual puede ser 0 o 1. Recuérdese que en caso de que H tenga valor 1 será porque la cabeza está en la celda cero, y entonces, la señal W_0 se bloqueará mediante la compuerta A_6 (ver figura 7.6) y todo el proceso acabará, es decir, la máquina de Turing se detendrá. 100
7.10. Configuración inicial de la unidad de control en el proceso movimiento de cabeza. Donde $\neg m$ es la negación de m y además m puede tomar el valor 0 o 1, según sea el movimiento que se va a realizar. 101
7.11. Configuración inicial en el proceso movimiento de cabeza de la i-ésima celda (lugar donde se localiza la cabeza de la máquina de Turing) y del resto de las celdas. El dato que se escribió recientemente en la i-ésima celda se denota por d', mientras que el símbolo $*$ representa a el valor del dato almacenado en el resto de las celdas. Tanto $*$ como d' pueden tomar valor 0 o 1, según sea el caso. 101
7.12. Señales y componentes involucrados en la unidad de control para el proceso de lectura del primer símbolo de la cadena w. 104
7.13. Señales y componentes involucrados en la celda cero para cuando se está yendo hacia la celda 1 para leer el primer símbolo de w. 105
7.14. Señales y compuertas involucradas en la i-ésima celda (i = 1) para el proceso de lectura del primer símbolo de w. En este caso el dato almacenado en el *flip-flop* D tiene valor 1. 106
7.15. Señales y componentes involucrados en la celda cero para cuando se está enviando el dato leído (d) por la señal DB_0 hacia la unidad de control. En este caso d = 1 porque la cadena de entrada es w = 10111. 107

7.16. Señales y componentes involucrados en el procesamiento de información. Se muestra desde el momento que la señales $\texttt{DB}_0$ y $\texttt{DP}_0$ llegan a la unidad de control hasta que las señales de escritura salen hacia la cinta. 108
7.17. Señales y componentes involucrados en el procesamiento de información. Se muestra desde el momento que se libera el retardo-4 hasta que las señales correspondientes al movimiento de cabeza salen hacia la cinta. 109
7.18. Señales y componentes involucrados en el procesamiento de información. Se muestra desde el momento que se libera el retardo-5 hasta que la señal $\texttt{R}_0$ sale hacia la cinta. 110
7.19. Señales y componentes involucrados en la celda cero para cuando se está yendo hacia la celda 1 para escribir el dato transferido a través de la señal $\texttt{DF}_0$. 111
7.20. Señales y componentes involucrados en la i-ésima celda ($i = 1$) para el proceso de escritura. En este caso se almacenará un 1 en el *flip-flop* `D`. 112
7.21. Señales y componentes involucrados en la celda cero para cuando se está por apagar la cabeza de la celda 1. 113
7.22. Señales y componentes involucrados en la i-ésima celda ($i = 1$) para apagar la señal del *flip-flop* `H` que indica que la cabeza está en ella. . . 115
7.23. Señales y compuertas involucradas en la i-ésima celda ($i = 2$) para encender la señal del *flip-flop* `H`. 116

Índice de figuras

2.1. Gráfica G con orden 13 y tamaño 24. La subgráfica G_2 es inducida mientras que G_1 no lo es. 6
2.2. La gráfica G es isomorfa con G_1 pero no con G_2. 7
2.3. Gráfica infinita, usualmente llamada *malla cuadrada*. 8
2.4. Digráfica D con ingrado mínimo $\delta^-(D) = 0$ e ingrado máximo $\Delta^-(D) = 3$. 8
2.5. Se muestran dos iteraciones del operador de clanes, $K(G)$ y $K^2(G)$, de la gráfica G. En este caso, G es K-convergente ya que $K^4(G) \cong K^3(G)$. 9
2.6. Diagrama de estados del AFD que reconoce a las cadenas binarias que representan a los números naturales. Nótese que el número cero es el único número que es representado por una cadena que empieza con el símbolo 0. El estado † es un estado muerto de $M_{\mathbb{N}}$. 12
2.7. Diagrama de estados del AFD que reconoce el lenguaje $L_1 = \{w \mid w = 1^a01^b0^c,\ a > 0, b \geq 0, c \geq 0\}$. Los estados muertos son omitidos. 13
2.8. Diagrama del AFD que acepta a la cadena formada por la convolución de $(011, 1000, 0000) \in (\Sigma_\diamond)^3$. Los estados muertos son omitidos. 18
2.9. Diagrama del AFD que acepta al lenguaje regular generado por la convolución de las cadenas de R_S. Los estados muertos son omitidos. 19
2.10. Diagrama del AFD que acepta al lenguaje regular generado por la convolución de las cadenas de $R'_S = \{(t, t+1, z) \mid t \in \mathbb{N} \text{ y } z \in \Sigma_\diamond\}$, la cual establece la cilindrificación de la relación automática R_S al agregar una tercera coordenada. De manera análoga se procede a cilindrificar a R_S si se agrega dos o más coordenadas. Los estados muertos son omitidos. 20
2.11. Diagrama del AFD que acepta al lenguaje regular generado por la convolución de las cadenas de $R''_S = \{(t+1, t) \mid t \in \mathbb{N}\}$, la cual establece la permutación de coordenadas de la relación automática R_S. Los estados muertos son omitidos. 21
2.12. (a) Gráfica G. (b) y (c) se muestran los diagramas de los autómatas M_V y M_E que reconocen a $V(G)$ y $\otimes E(G)$, respectivamente; donde $\Sigma = \{0, 1, \ldots, 8\}$ (en ambos casos, los estados muertos son omitidos). Las flechas del autómata M_E están ordenadas de manera lexicográfica de arriba hacia abajo. 23

2.13. (a) Gráfica infinita G'. Mientras que en (b) y (c) se muestran los diagramas de los autómatas que reconocen a los vértices y aristas de G', respectivamente (en ambos, los estados muertos son omitidos). . . 23

2.14. Esquema de una máquina de Turing. La cabeza es representada mediante la flecha que está entre la unidad de control y la cinta. 24

2.15. Diagrama del AFD que reconoce a las transiciones $s \vdash s'$ de la máquina de Turing M_P (ver ejemplo 2.6.3) por medio de las convoluciones $\otimes(s, s')$. Se llegará a algún estado final (q_1 o q_2) sólo cuando $s \vdash s'$; de lo contrario se alcanzará al estado muerto, el cual es omitido en el diagrama. 28

2.16. Diagrama del AFD genérico que reconoce a las transiciones $s_i \vdash s_j$ que se generan con una máquina de Turing $M = (Q, \Sigma, \Gamma, \delta, p_0, \llcorner, F)$ por medio de las convoluciones $\otimes(s_i, s_j)$. Los símbolos a, b y $c \in \Gamma$; mientras que p y $q \in Q$. Los estados muertos son omitidos en el dibujo. 29

2.17. Diagrama de bloques del algoritmo `X` que resuelve el problema del paro en la demostración por contradicción. 32

3.1. Parte del autómata M_3 que reconoce a las configuraciones con estampa de tiempo, cuando $|t| = |t + 1|$. Esta está formada por una copia de los autómatas 2.9 y 2.16 (denotados por $M_1 = (Q_1, (\Sigma_{1\diamond})^2, q_1^0, \delta_1, F_1)$ y $M_2 = (Q_2, (\Sigma_{2\diamond})^2, q_2^0, \delta_2, F_2)$, respectivamente). Sólo el estado q_1^0 es estado inicial en M_3; de igual manera, sólo los estados del conjunto F_2 son finales. 36

3.2. Parte del autómata M_3 que reconoce a las configuraciones con estampa de tiempo, cuando $|t| < |t + 1|$. Esta está formada por la misma copia de M_1 de la parte anterior (ver imagen 3.1) y por $n = |(\Gamma \cup Q)_\diamond|$ copias del autómata M_2 (denotadas por $M_2[s_i]$, $s_i \in (\Gamma \cup Q)_\diamond$. Solamente el estado q_1^0 es estado inicial en M_3; de igual manera sólo los estados del conjunto $\{F_2 \times (\Gamma \cup Q)_\diamond\}$ son finales. Las flechas curvas en color negro permiten transitar entre las n copias de M_2 para simular la lectura de $\otimes(s, s')$, si $ts \vdash t's'$. 37

4.1. Construcción de $\gamma_0(M)$. Los vértices de la forma $4d$, $5d$ y $6d$ se omiten en (c). 43

4.2. Dos fragmentos de la gráfica $\gamma_0(M)$ cuando d es no terminal (de las dos posibles maneras). Los vértices y aristas en color gris pueden estar o no estar presentes en la cinta. En (a) se presenta el caso $d' \vdash d_1$ y en (b) el caso $d_1 \vdash d'$. Para ambos casos, el triángulo extendido del triángulo azul está sombreado de color rosa y tiene como ápice al vértice $2d'$. Los vértices de la forma $4d$, $5d$ y $6d$ son omitidos en los dibujos. 44

4.3. Dos fragmentos de la gráfica $\gamma_0(M)$ cuando d es terminal (de las dos posibles maneras). Los vértices y aristas en color gris pueden estar o no estar presentes en la cinta. En el caso (a), el triángulo extendido del triángulo azul está sombreado de color rosa y tiene como ápices a los vértices $3d'$ y $2d'$. Los vértices de la forma $4d$, $5d$ y $6d$ son omitidos en los dibujos. 45
4.4. (a) y (b) muestra a $\gamma(M, d)$ y $\gamma(M, d')$, respectivamente. Los vértices de la forma $4d$, $5d$ y $6d$ son omitidos en los dibujos. 46
4.5. Tipos de triángulos de $\gamma_0(M)$. Los vértices de la forma $4d$, $5d$ y $6d$ son omitidos del dibujo. 48

5.1. (a) X_4, (b) Y_4. 56
5.2. (a) $\mathfrak{A}_1 = A_4$, (b) $\mathfrak{B}_1 = B_4$. 58

6.1. (a) Bloque de construcción básico. (b) Bloque de construcción básico sin vértices dominados. 62
6.2. Canal. 63
6.3. Las subgráficas en color rojo representan a las perturbaciones locales (fotones) de los canales, las cuales se mueven (en este caso, de izquierda a derecha) a causa del operador de clanes. Si G es la gráfica que se muestra en (a), entonces las gráficas de (b) y (c) son $K^2(G)$ y $K^4(G)$, respectivamente. 64
6.4. Gráfica que representa la configuración de la compuerta `OR` cuando en ambas entradas, A y B, hay un cero (ausencia de fotón). Dado que la gráfica es K^2-invariante en la salida Q no se presentará ninguna perturbación. 65
6.5. La gráfica G en (a) representa la configuración de la compuerta `OR` cuando en A entra un 1 y en B un 0. Las gráficas de (b) hasta (f) son $K^2(G)$ a $K^{10}(G)$, respectivamente. De (b) a (f) se muestra el traslado gradual del fotón hacia la salida Q. 65
6.6. La gráfica G en (a) representa la configuración de la compuerta `OR` cuando en A entra un 0 y en B un 1. Las gráficas de (b) hasta (f) son $K^2(G)$ a $K^{10}(G)$, respectivamente. De (b) a (f) se muestra el traslado gradual del fotón hacia la salida Q. 66
6.7. La gráfica G en (a) representa la configuración de la compuerta `OR` cuando en A y B entra un 1. Las gráficas de (b) hasta (f) son $K^2(G)$ a $K^{10}(G)$, respectivamente. Obsérvese que en las gráficas de (a) hasta (d) hay dos fotones, los cuales a partir de (e) se fusionan en uno solo para salir por Q. 67
6.8. Divisor de señal con entrada en A y salidas en B y C. Nótese este componente es el reflejo horizontal de la gráfica que representa a la compuerta `OR`. 67

6.9. En (a) se muestra la gráfica G que representa al divisor de señal con un fotón en la entrada A. Las gráficas de (b) hasta (f) son $K^2(G)$ a $K^{10}(G)$, respectivamente. En la iteración $K^8(G)$, inciso (e), se observan dos fotones en canales distintos debido a que el operador de clanes duplica el fotón original. En las siguientes iteraciones, estos fotones continuarán trasladándose por los canales (ver inciso (f)) hasta llegar a las salidas B y C. 68

6.10. Empalme. Duplica el fotón entrante permitiendo que los dos fotones se distribuyan hacia los demás canales. Por ejemplo, si el fotón entra por A entonces saldrá un fotón por B y otro por C (ver figura 6.11). . 68

6.11. En (a) se muestra la gráfica G que representa al empalme con un fotón en la entrada A. Las gráficas de (b) hasta (f) son $K^2(G)$ a $K^{10}(G)$, respectivamente. En la iteración $K^4(G)$, inciso (e), se observa que el fotón de entrada en A es duplicado por el operador de clanes. En las siguientes dos iteraciones, $K^6(G)$ y $K^8(G)$, los fotones se trasladan hacia los canales B y C. 69

6.12. En (a) se muestra la gráfica G que representa al empalme con un fotón en la entrada B. Las gráficas de (b) hasta (f) son $K^2(G)$ a $K^{10}(G)$, respectivamente. En la iteración $K^4(G)$, inciso (e), se observa que el fotón de entrada en B es duplicado por el operador de clanes. En las siguientes dos iteraciones, $K^6(G)$ y $K^8(G)$, los fotones se trasladan hacia los canales A y C. 70

6.13. En (a) se muestra la gráfica G que representa al empalme con un fotón en la entrada C. Las gráficas de (b) hasta (f) son $K^2(G)$ a $K^{10}(G)$, respectivamente. En la iteración $K^4(G)$, inciso (e), se observa que el fotón de entrada en B es duplicado por el operador de clanes. En las siguientes dos iteraciones, $K^6(G)$ y $K^8(G)$, los fotones se trasladan hacia los canales A y B. 71

6.14. La gráfica G en (a) representa la configuración de la compuerta `AND` cuando en A y B entra un 0. Las gráficas de (b) hasta (h), que son $K^2(G)$ a $K^{14}(G)$ respectivamente, tienen una gran cantidad de fotones indeseados para simular el comportamiento de la compuerta `AND`. En particular el orden de la gráfica en (f) es de 93. 73

6.15. La gráfica G en (a) representa la configuración de la compuerta `AND` cuando en A entra un 1 y en B un 0. Las gráficas de (b) hasta (h) son $K^2(G)$ a $K^{14}(G)$, respectivamente. En particular el orden de la gráfica en (f) es de 93. 74

6.16. La gráfica G en (a) representa la configuración de la compuerta `AND` cuando en A entra un 0 y en B un 1. Las gráficas de (b) hasta (h) son $K^2(G)$ a $K^{10}(G)$, respectivamente. En particular el orden de la gráfica en (f) es de 94. 75

6.17. La gráfica G en (a) representa la configuración de la compuerta AND cuando en A y en B entra un 1. Las gráficas de (b) hasta (h) son $K^2(G)$ a $K^{10}(G)$, respectivamente. En el canal de salida Q de la gráfica (f) se está produciendo el fotón extra requerido por la función AND, por eso mismo, el orden de esta gráfica es de 95. 76
6.18. Esquema de la hipotética compuerta NOT. Éste consta de una caja negra que tiene conectado un canal de entrada y uno de salida. Las gráficas H_1 y H_3, que representan a la compuerta NOT cuando respectivamente entra 1 y un 0, después de n iteraciones del operador de clanes se transforman en H_2 y H_4, respectivamente. 77
6.19. Canal doble. 78
6.20. Segunda codificación. En (a) y (b) se muestra la codificación respectivamente del 1 y del 0 usando canales dobles. 78
6.21. Compuerta NOT. Las líneas etiquetadas con A y $\bar{\text{A}}$ representan respectivamente al canal superior e inferior del canal doble de entrada en la compuerta NOT; mientras que las líneas Q y $\bar{\text{Q}}$ representan al canal superior e inferior del canal doble de salida. 79
6.22. Compuerta AND. Está formada por la compuerta OR y AND de la primera codificación. La compuerta AND tiene el símbolo "?" ya que se está considerando la posible existencia de la compuerta AND-limpia de la primera codificación. 80

7.1. Esquema del diseño digital que simula la máquina de Turing. Los rectángulos alargados representan a las celdas de memoria y el círculo relleno marca la presencia de la cabeza en la celda correspondiente (de lo contrario, el círculo vacío significa que la cabeza no está en ella). La celda cero (localiza a lado de la unidad de control) no almacena ningún dato. 82
7.2. *Flip-flop* SR. 83
7.3. *Flip-flop* D. 84
7.4. (a) Símbolo gráfico del *flip-flop* SR. (b) Símbolo gráfico del *flip-flop* D. 84
7.5. Diseño de la unidad de control (UC) de la máquina de Turing definida en la tabla 7.3. 85
7.6. Diseño de la celda cero de memoria para cualquier máquina de Turing. 90
7.7. Diseño de la i-ésima celda de memoria ($i > 0$) para cualquier máquina de Turing. 91
7.8. Sucesión cíclica de los procesos que se realizan en la simulación de la ejecución de la máquina de Turing. 93

Capítulo 1

Introducción

El *operador de clanes* (K) transforma a una gráfica G en su *gráfica de clanes* $K(G)$ mediante la intersección de sus clanes (subgráficas completas maximales de G): los vértices son los clanes y dos de estos vértices son adyacentes si y sólo si los clanes que los representan comparten por lo menos un vértice en la gráfica G. Las gráficas iteradas de clanes son definidas por $K^n(G) = K(K^{n-1}(G))$ donde $K^0(G) = G$. Entonces, si para alguna $n \neq m$ sucede que $K^n(G) \cong K^m(G)$, se dice que G es *clan-convergente* (K-convergente), de lo contrario se dice que G es *clan-divergente* (K-divergente). El *clan-comportamiento* (K-comportamiento) de G indica si ésta es K-convergente o K-divergente mediante la aplicación iterada del operador de clanes sobre G.

El operador de clanes es ampliamente considerado como uno de los operadores iterados más complejos en gráficas [52]. Además, la caracterización de la K-convergencia ha resistido todos los intentos por 49 años desde que la noción de las gráficas iteradas de clanes fue introducida en [24]. Tampoco se ha tenido un avance considerable para demostrar que la clan-convergencia es decidible, es decir, no se ha demostrado la existencia de un algoritmo (programa computacional que siempre se detiene) que decida la clan-convergencia para la clase de gráficas simples finitas, de hecho, este problema aún se mantiene abierto. La indecibilidad de la clan-convergencia se ha sugerido en distintos lugares, comenzando en [48] e incluyendo a [53]. La primera declaración explícita sobre este tema apareció en [44] y la primera declaración explícita en una publicación formal en [40].

En esta investigación se presenta el primer resultado de indecibilidad para la clan-convergencia con una relajación en las gráficas que contempla originalmente el problema, las cuales son gráficas simples finitas. Específicamente se relajó la condición de finitud tan poco como fuera posible, por lo cual, se consideró sólo al conjunto de gráficas (posiblemente infinitas pero) finitamente presentables, localmente finitas con grado acotado, con a lo más un vértice dominado y cuasi-clan-Helly (gráficas que se convierten en clan-Helly después de eliminar un vértice). Estableciendo condiciones más estrictas, se restringió aún más la clase de *gráficas automáticas*, cuyos vértices y

aristas pueden ser reconocidos por autómatas finitos deterministas (ver sección 2.3). Los autómatas son los modelos computacionales más simples, por lo que las gráficas automáticas no sólo tienen algoritmos para determinar los vértices y aristas, sino que estos algoritmos son de la forma más simple.

Las gráficas automáticas y las *estructuras automáticas* [2,4,30,31,54] tienen fuertes propiedades de decibilidad heredadas de la teoría de autómatas. De hecho, la teoría de primer orden (conjunto de expresiones formadas con variables, conjunciones, disyunciones, negaciones y con los cuantificadores $\forall$ y $\exists$) de cualquier estructura automática es decidible, es decir, que existe un algoritmo que decide la veracidad de cualquier proposición de primer orden que hable sobre esa estructura [30]. Así mismo, la teoría de primer orden expandida con la generalización de ciertos cuantificadores lógicos son aún decidibles [54] e incluso algunos fragmentos de la teoría del segundo orden son también decidibles [31].

Considerando a las gráficas automáticas, se probó en el capítulo 4 que la K-convergencia es algorítmicamente indecidible para esta clase de gráficas, incluso si éstas además son cuasi-clan-Helly, con grado acotado y con a lo más un vértice dominado. La prueba de este resultado se efectúa mediante una reducción desde el problema del paro, que es bien conocido como un problema indecidible. Para esta reducción se tomó en cuenta que las configuraciones (o descripciones instantáneas) de una máquina de Turing (MT) tendrían concatenada una cadena binaria t (del lado izquierdo) para representar la estampa de tiempo de la configuración, de modo que si $ts \vdash t's'$ es porque $t' = t + 1$ y $s \vdash s'$, donde el símbolo $\vdash$ indica la relación de transición entre configuraciones de la máquina de Turing. Dichas configuraciones son definidas y explicadas en el capítulo 3.

El resultado del párrafo anterior en conjunto con el teorema 2.5.2 implica que la propiedad de la clan-convergencia no es expresable en lenguaje de primer orden para gráficas automáticas, ya que de lo contrario, la clan-convergencia sería decidible. Extendiendo este resultado de inexpresibilidad, en el capítulo 5, se prueba que además la propiedad de la clan-divergencia no es expresable en lenguaje de primer orden para la clase de gráficas finitas. El resultado plantea una estrategia ganadora para el juego de k-rondas de Ehrenfeucht-Fraïssé ocupando dos familias de gráficas cuasi-isomorfas.

Por otra parte, en el año 2001, Medianis [44] planteó si el operador de clanes es capaz de simular a una máquina de Turing dada la dinámica de dicho operador. Abordando esta inquietud, en el capítulo 6 se muestra el primer acercamiento de la simulación de circuitos digitales mediante gráficas de clanes. En él se presenta la simulación de portadores de señal (fotones), cables, divisores, empalmes y la compuerta `OR`, sobre lo que fue llamado primera codificación. También muestra la compuerta `NOT` sobre lo que fue llamado segunda codificación. Cabe mencionar que la simulación completa depende de la existencia de la compuerta `AND`-limpia sobre la primera codificación, la cual fue buscada mediante algoritmos genéticos pero no se tuvo algún éxito. Sin embargo, se encontró a la `AND`-sucia, llamada así porque además de presentar el comportamiento de la compuerta lógica `AND`, genera otros

vértices más en la aplicación iterada del operador de clanes.

En el apéndice A se muestran los códigos diseñados en el software YAGS [9] que permiten explorar y visualizar a las compuertas `OR` y `AND`-sucia sobre la primera codificación. La implementación de ambas contempla el diseño de otros componentes básicos como son: portadores de señal, canales, divisores y empalmes. Para visualizar los componentes se detalla la creación de dos archivos necesarios (gadgetData.g y gadgets.g), así como el código de exploración y de compilación en YAGS.

Además, usando los componentes digitales (simulados con gráficas de clanes) y con la suposición de la existencia de la gráfica que simula a la compuerta `AND`-limpia, se muestra en el capítulo 7 cómo simular a cualquier máquina de Turing estrictamente binaria (cuyo alfabeto contiene sólo al 0 y al 1, y que también omite al símbolo de espacio "␣"). Específicamente se explica cómo sería el diseño de las partes que conforman a una máquina de Turing, es decir, a la unidad de control y a la cinta infinita de memoria. La simulación de esta última contempla dos tipos de celdas de memoria: la celda cero (para considerar el desbordamiento de la cabeza de la MT) y las i-ésimas celdas (para almacenar la cadena w de entrada de la MT).

Los resultados de esta investigación dieron lugar a tres artículos en revistas indexadas: [11], [12] y [13], donde dos de ellos están en revistas JCR. Además permitió el enriquecimiento de funciones en YAGS [9] para el desarrollo de los componentes digitales del capítulo 6 y la creación de [10] como referencia complementaria para visualizar dinámicamente a dichos componentes.

Capítulo 2

Antecedentes

En este capítulo se mencionan conceptos y resultados de teoría previamente desarrollada que es necesaria para el desarrollo de los capítulos siguientes. Principalmente se abordan temas sobre la teoría de la computación pero también de la teoría de gráficas haciendo énfasis en las gráficas de clanes. La combinación de estos conceptos sustentan la investigación plasmada a lo largo de este documento.

2.1. Teoría de gráficas

Una *gráfica* G es un par ordenado (V, E) donde $V = V(G)$ es un conjunto finito no vacío de *vértices* y $E = E(G)$ es un conjunto de *aristas* (pares no ordenados de V). Se denotará por $e = uv$ a la arista formada por el par de vértices $u, v \in V$; además se dirá que u y v son *incidentes* con la arista e y viceversa. Dos vértices que son incidentes en una arista en común son *adyacentes*. Si v_1 y v_2 son vértices adyacentes en G, entonces se dice que v_1 y v_2 son *vecinos*; mientras que la *vecindad* de v_1, $N(v_1)$, consiste en todos los vecinos de v_1 y la *vecindad cerrada* de v_1 es $N[v_1] = N(v_1) \cup \{v_1\}$. El número de vértices y aristas de G son llamados, respectivamente, *orden* y *tamaño* de G. El *grado* de un vértice v de G, denotado por $d_G(v)$, es el número de aristas incidentes con v. Se denota por $\delta(G)$ y $\Delta(G)$ al mínimo y máximo grado de los vértices de G, respectivamente [6].

Las gráficas pueden ser representadas geométricamente: cada vértice es indicado por un punto o círculo pequeño y cada arista por un segmento de línea o curva uniendo los puntos. Su representación geométrica ayuda a entender muchas de sus propiedades. En la figura 2.1 se muestra una gráfica $G = (V, E)$ con orden 13 y tamaño 24, donde: $V = \{v_1, v_2, \ldots, v_{13}\}$ y $E = \{v_1v_2, v_1v_4, v_1v_7, v_1v_{10}, v_1v_{12}, v_1v_{13}, v_2v_3, v_3v_4, v_3v_5, v_4v_5, v_4v_7, v_4v_{10}, v_5v_6, v_6v_7, v_7v_8, v_7v_9, v_7v_{10}, v_9v_{10}, v_9v_{11}, v_{10}v_{11}, v_{10}v_{12}, v_{10}v_{13}, v_{11}v_{12}, v_{12}v_{13}\}$, además el grado del vértice v_4, el grado mínimo y máximo de G son respectivamente, $d_G(v_4) = 5$, $\delta(G) = 1$ y $\Delta(G) = 7$.

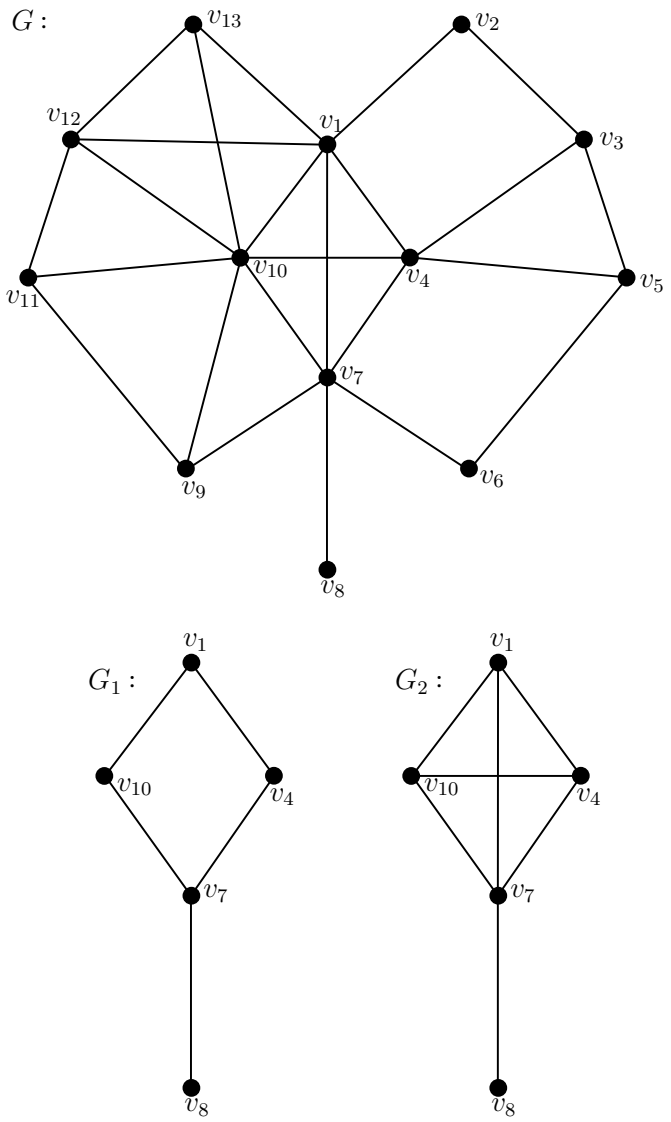


Figura 2.1: Gráfica G con orden 13 y tamaño 24. La subgráfica G_2 es inducida mientras que G_1 no lo es.

Un *camino* es una secuencia finita de vértices donde cada par de vértices consecutivos en la secuencia están conectados por una arista. El número de aritas en un camino se llama *longitud* del camino. En la gráfica G_2 de la figura 2.1, la secuencia v_1, v_4, v_7, v_8 representa un camino entre el vértice v_1 y v_8 con longitud 3.

Dada una gráfica G, x, $y \in V$ y los subconjuntos A, $B \subseteq V$, se define a la distancia entre x y y como la longitud de la trayectoria más corta que hay entre dichos vértices y se denota por $d_G(x, y)$. Así mismo se define a $d_G(x, A) = min\{d_G(x, y) : y \in A\}$ y a $d_G(A, B) = min\{d_G(x, y) : x \in A,\ y \in B\}$. Por ejemplo, si se considera la gráfica G

de la figura 2.1 y a los conjuntos A, $B \in V(G)$, tal que $A = \{v_1, v_{10}, v_{11}, v_{12}, v_{13}\}$ y $B = \{v_4, v_5, v_6v_7\}$, entonces $d_G(v_6, v_9) = 2$, $d_G(v_8, A) = 2$ y $d_G(A, B) = 1$.

Dadas dos gráficas G y H se dice que son *isomorfas* ($G \cong H$) si existe una correspondencia uno a uno entre sus conjuntos de vértices que preserva la adyacencia; es decir, dos vértices v_1 y v_2 en G son adyacentes si y sólo si, los correspondientes vértices u_1 y u_2 de H son adyacentes. En la figura 2.2 se puede observar que la gráfica G es isomorfa con G_1, pero no lo es con G_2. Esto último sucede por que los vértices $a, b, c, d \in G$ tienen correspondencia, respectivamente, con los vértices $v_1', v_2', v_4', v_3' \in G_2$, pero el vértice v_2' tiene una adyacencia diferente en G_2.

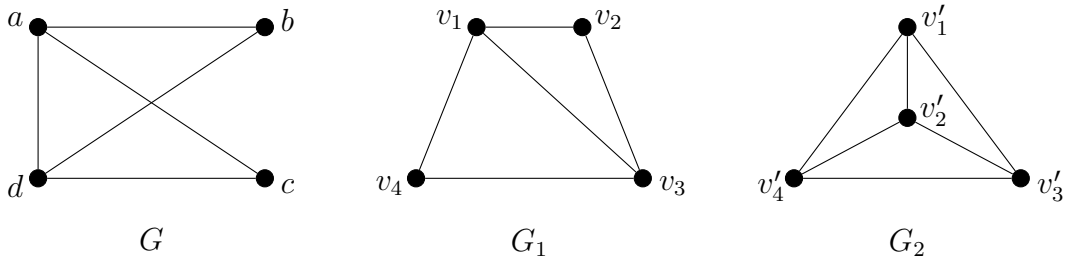


Figura 2.2: La gráfica G es isomorfa con G_1 pero no con G_2.

Una gráfica H es *subgráfica* de una gráfica G si $V(H) \subseteq V(G)$ y $E(H) \subseteq E(G)$. Para cualquier subconjunto S de $V(G)$, la *subgráfica inducida* $G[S]$ es la subgráfica de G que tiene a S como conjunto de vértices y donde dos vértices son adyacentes en $G[S]$ si y solo si son adyacentes en G. Por ejemplo, en la figura 2.1, las gráficas G_1 y G_2 son subgráficas de G y G_2 es una subgráfica inducida de G pero G_1 no lo es.

Existen distintas familias de gráficas que son de gran utilidad en la teoría de gráficas. Por ejemplo, las gráficas *completas* son gráficas donde cada par de vértices son adyacentes. Una *trayectoria* también es una gráfica cuyos vértices pueden ser ordenados en una línea de manera consecutiva, donde dos vértices son adyacentes si y sólo si estos son consecutivos. Se dice que una gráfica es *conexa* si para toda partición de sus vértices en conjuntos no vacíos P_1 y P_2, existe una arista que una a P_1 con P_2, de lo contrario se dice que es *disconexa*.

Una gráfica es *finita* si el conjunto de vértices es finito. La gráfica que no tiene vértices, y por lo tanto que no tiene aristas, es llamada gráfica *vacía*; mientras que la gráfica *trivial* es aquella que sólo tiene un vértice y la gráfica *discreta* es aquella que tiene un conjunto de aristas vacío.

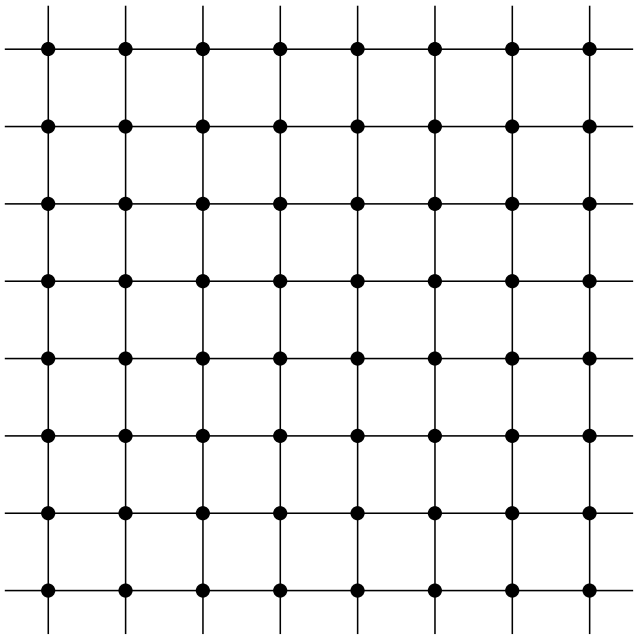


Figura 2.3: Gráfica infinita, usualmente llamada *malla cuadrada*.

Una gráfica es *infinita* si el conjunto de vértices es infinito. Si el conjunto de vértices de una gráfica infinita G es *numerable* se dice que G es numerable. La figura 2.3 muestra un ejemplo de una gráfica infinita que es resultante del producto cartesiano de dos trayectorias infinitas. Con frecuencia las propiedades y/o fundamentos que son válidos para las gráficas finitas se pueden aplicar directamente en gráficas infinitas.

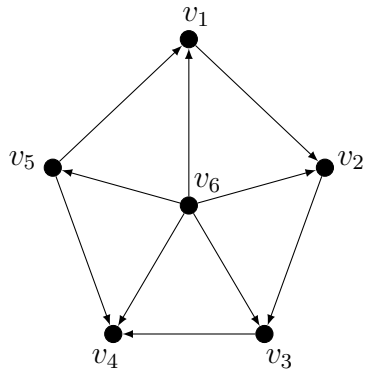


Figura 2.4: Digráfica D con ingrado mínimo $\delta^-(D) = 0$ e ingrado máximo $\Delta^-(D) = 3$.

Por otra parte, una *gráfica dirigida* (o digráfica), también denotada por $D = (V, E)$, consiste en un conjunto de vértices V y un conjunto de pares ordenados

de vértices E, llamados *flechas*. Si la flecha $uv \in E$, entonces la flecha va de u a v y es incidente con los vértices u y v. El *exgrado* de un vértice $v \in D$, $d_D^+(v)$, es el número de flechas que salen de él, mientras que el *ingrado* de v, $d_D^-(v)$, es el número de flechas que llegan a él. El mínimo ingrado y mínimo exgrado de D se denota por $\delta^-(D)$ y $\delta^+(D)$, respectivamente. Así mismo, el máximo ingrado y máximo exgrado de D son denotados respectivamente por $\Delta^-(D)$ y $\Delta^+(D)$. En la figura 2.4 se muestra a la digráfica $D = (V, E)$, donde $V = \{v_1, v_2, \ldots, v_6\}$ y $E = \{v_1v_2, v_2v_3, v_3v_4, v_5v_1, v_5v_4, v_6v_1, v_6v_2, v_6v_3, v_6v_4, v_6v_5\}$, con $\delta^-(D) = 0$, $\delta^+(D) = 0$, $\Delta^-(D) = 3$ y $\Delta^+(D) = 5$, mientras que el ingrado y exgrado del vértice v_5 son respectivamente, $\delta_D^-(v_5) = 1$ y $\delta_D^+(v_5) = 2$.

2.2. Gráficas de clanes

Dada una gráfica G, se dice que un *clan* es una subgráfica completa maximal de G. El *operador de clanes* transforma a G en la *gráfica de clanes*, $K(G)$, la cual se genera con la intersección de los clanes de G: los vértices son los clanes y dos de ellos son adyacentes si y sólo si comparten por lo menos un vértice. Las *gráficas iteradas de clanes* están definidas por $K^0(G) = G$ y $K^n(G) = K(K^{n-1}(G))$. Si existe $n \neq m$ tal que $K^n(G) \cong K^m(G)$ se dice que G es *clan-convergente* (o K-*convergente*); de lo contrario G es *clan-divergente* (o K-*divergente*). El comportamiento dinámico de G bajo la aplicación iterada del operador de clanes es llamado el *clan-comportamiento* (ó K-*comportamiento*) de G, por lo que determinar el K-comportamiento de G se refiere a determinar si G es K-convergente o K-divergente. En la figura 2.5 se muestra un ejemplo de una gráfica G que es K-convergente ya que $K^3(G)$ es la gráfica trivial, entonces $K^4(G) \cong K^3(G)$.

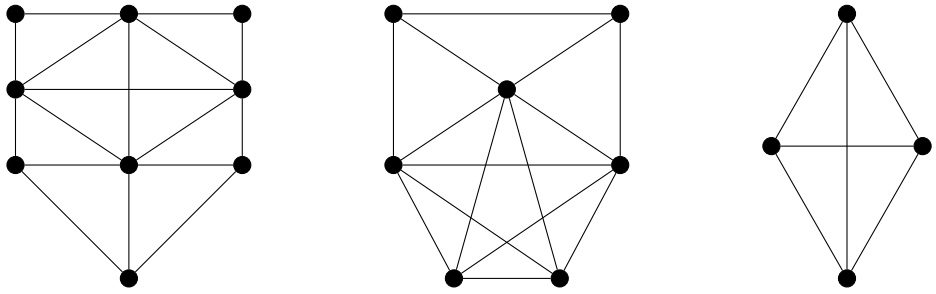


Figura 2.5: Se muestran dos iteraciones del operador de clanes, $K(G)$ y $K^2(G)$, de la gráfica G. En este caso, G es K-convergente ya que $K^4(G) \cong K^3(G)$.

Las gráficas iteradas de clanes fueron introducidas en el año de 1972 por Hedetniemi y Slater [24] y desde entonces las gráficas iteradas de clanes han sido estudiadas

extensivamente. Algunos resultados se pueden consultar en [42, 52, 57]. Específicamente la K-convergencia puede ser revisada en [1, 5, 7, 19–21, 24, 26, 32, 38, 39, 51] y la K-divergencia en [17, 19, 20, 32–37, 40, 41, 49, 50].

Una gráfica G es *K^n-invariante* si $K^n(G) \cong G$. En particular, G es K^2-invariante cuando $K^2(G) \cong G$. Cuando $K^n(G)$ es isomorfa a la gráfica trivial para alguna n, se dice que G es K-nula.

Se dice que un vértice x es *dominado* si existe algún vértice $y \neq x$ tal que $N[x] \subseteq N[y]$. Una gráfica G es un *cono* si ésta contiene un *ápice*, el cual es un vértice adyacente a todos los otros vértices de G. Dado un triángulo $T = \{x, y, z\}$ de G, su *triángulo extendido* es $\hat{T} = \{u \in G : |N(u) \cap T| \geq 2\}$.

Teorema 2.2.1 (Dragan [15], Szwarcfiter [57, 58]). *Una gráfica G es clan-Helly si y sólo si para todo triángulo $T \in G$, su triángulo extendido $\hat{T}$ es un cono.*

Dado un vértice $x \in G$, su *estrella* es $x^* = \{q \in K(G) : x \in q\}$. Las estrellas pueden o no pueden ser *clanes de clanes* de G (es decir, clanes de $K(G)$ y vértices de $K^2(G)$) ya que pueden no ser maximales. Los clanes de clanes que no son estrellas se llaman *corbatas*. Si las estrellas $x^*, y^* \in K^2(G)$, entonces tienen la propiedad de $x^* \simeq y^*$ en $K^2(G)$ si y sólo si $x \simeq y \in G$ ya que ambas condiciones son equivalentes a $x, y \in q$ para algún $q \in K(G)$.

Teorema 2.2.2 (Escalante [17]). *Si una gráfica G es clan-Helly sin vértices dominados, entonces $G \cong K^2(G)$. Además, el isomorfismo está dado por $x \mapsto x^*$.*

Ambos teoremas son para gráficas finitas, pero es fácil verificar que sus pruebas son válidas para gráficas infinitas. El teorema 2.2.2 es válido para gráficas infinitas y el teorema 2.2.1 es válido para gráficas localmente finitas (es decir, que el grado de todo vértice es finito).

Se dice que una gráfica G es *desmantelable a H en un sólo paso* ($G \xrightarrow{\#} H$) si G contiene una subgráfica inducida $H_0 \cong H$ tal que todo vértice en $G \setminus H_0$ es dominado por algún vértice en H_0, es decir, cuando $\forall x \in G \setminus H_0\ \exists y \in H_0, N_G[x] \subseteq N_G[y]$.

Teorema 2.2.3 (Frías, Neumann-Lara, Pizaña. Teorema 3 en [20]). *Si $G \xrightarrow{\#} H$ entonces $K(G) \xrightarrow{\#} K(H)$.*

Teorema 2.2.4 (Frías, Neumann-Lara, Pizaña. Teorema 5 en [20]). *Si G es desmantelable a H, entonces G y H tienen el mismo K-comportamiento. En particular, si x es un vértice dominado de G, $G - \{x\}$ tiene el mismo $K-$comportamiento.*

Las gráficas clan-Helly dan pie al siguiente teorema de Escalante:

Teorema 2.2.5 (Escalante [18]). *Sea G una gráfica clan-Helly y H cualquier subgráfica inducida minimal de G que satisfaga que $\forall x \in G\ \exists y \in H$ tal que y domina a x. Entonces, $K^2(G) \cong H$. En particular, toda gráfica clan-Helly es K-convergente de periodo 1 o 2.*

2.3. Teoría de autómatas

Un alfabeto Σ es cualquier conjunto finito, donde los miembros del alfabeto son llamados *símbolos*. Una *cadena* w sobre Σ es una secuencia finita de símbolos del alfabeto Σ. La *longitud* de w, denotada por $|w|$, es el número de símbolos que w contiene. La cadena de longitud cero es llamada *cadena vacía* y usualmente denotada o por ε, con $|\varepsilon| = 0$. Si w tiene longitud n se escribe $w = a_1 a_2 \dots a_n$ donde cada $a_i \in \Sigma$ y $w[i] = a_i$ denota al i-ésimo símbolo de w, tal que $1 \leq i \leq n$.

El conjunto de todas las cadenas sobre Σ es denotado por Σ^*. Un lenguaje L sobre Σ es cualquier conjunto $L \subseteq \Sigma^*$. Por ejemplo, $\Sigma = \{0, 1\}$, entonces $\Sigma^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, \dots\}$. El conjunto vacío, $\varnothing$, y el conjunto formado por la cadena vacía $\{\varepsilon\}$ también se consideran como lenguajes. Hay que notar que son dos lenguajes distintos, el primero no tiene elementos mientras el segundo tiene a uno, a la cadena vacía.

Un *autómata finito* (AFD) es una 5-tupla $M = (Q, \Sigma, \delta, q_0, F)$ donde:

1. Q es un conjunto finito de estados,
2. Σ es el alfabeto,
3. $\delta : Q \times \Sigma \longrightarrow Q$ es la función de transición,
4. $q_0 \in Q$ es el estado de inicio, y
5. $F \subseteq Q$ es el conjunto de estados de aceptación ó estados finales.

Si A es el conjunto de todas las cadenas que reconoce M, $A = L(M)$, se dice que A es reconocido por M. Un lenguaje que es reconocido por un autómata finito es llamado *lenguaje regular*.

Ejemplo 2.3.1. *Sea* $M_{\mathtt{N}} = (Q, \Sigma, \delta, q_0, F)$ *el autómata que reconoce al lenguaje regular* $\mathtt{N}$*, donde* $\mathtt{N}$ *es el conjunto de todas la cadenas binarias que representan a los números naturales* $\mathbb{N}$*. Entonces,* $M_{\mathtt{N}}$ *es definido de la siguiente manera:*

1. $Q = \{q_0, q_1, q_2, \dagger\}$,
2. $\Sigma = \{0, 1\}$,
3. δ :

	0	1
q_0	q_1	q_2
q_1	$\dagger$	$\dagger$
q_2	q_2	q_2
$\dagger$	$\dagger$	$\dagger$

4. $q_0 \in Q$ *es el estado inicial, y*

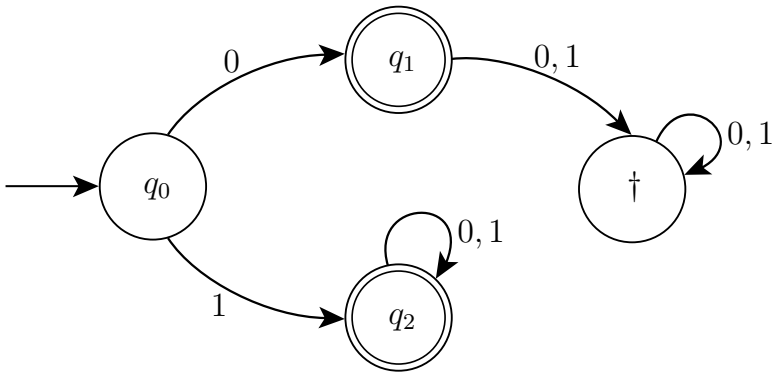


Figura 2.6: Diagrama de estados del AFD que reconoce a las cadenas binarias que representan a los números naturales. Nótese que el número cero es el único número que es representado por una cadena que empieza con el símbolo 0. El estado † es un estado muerto de $M_{\mathbb{N}}$.

5. $F = \{q_1, q_2\}$.

Obsérvese que si una cadena $w \notin \mathbb{N}$, la función de transición del autómata $M_{\mathbb{N}}$ llegará al estado denotado por † el cual es nombrado *estado muerto*. En la figura 2.6 se muestra el *diagrama de estados* de $M_{\mathbb{N}}$, donde los círculos denotan a los estados del autómata y las flechas representan las transiciones de estados de la función δ. Usualmente en la práctica del diseño de estos diagramas no es común mostrar las transiciones que hay hacia los estados muertos, por lo cual se da por entendido que la ausencia de ciertas flechas en el diagrama indican que existen transiciones a los estados muertos. Siguiendo esta práctica de aquí en adelante los diagramas de estados de los autómatas se mostrarán sin estos estados, tal como se puede ver en el siguiente ejemplo:

Ejemplo 2.3.2. *Sea* $M_1 = (Q, \Sigma, \delta, q_0, F)$ *el autómata (figura 2.7) que reconoce al lenguaje* $L_1 = \{w \mid w = 1^a01^b0^c,\ a > 0, b \geq 0, c \geq 0\}$*:*

1. $Q = \{q_0, q_1, q_2, q_3, \dagger\}$,

2. $\Sigma = \{0, 1\}$,

3. δ :

	0	1
q_0	†	q_1
q_1	q_2	q_1
q_2	q_3	q_2
q_3	q_3	†
†	†	†

4. $q_0 \in Q$ *es el estado inicial, y*

5. $F = \{q_2, q_3\}$.

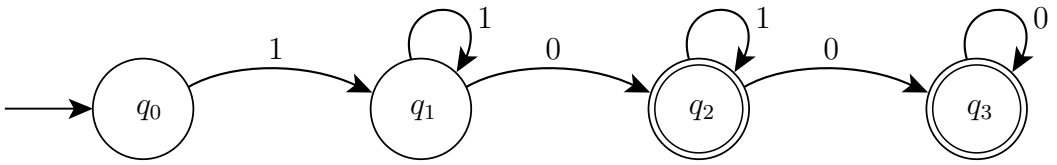


Figura 2.7: Diagrama de estados del AFD que reconoce el lenguaje $L_1 = \{w \mid w = 1^a01^b0^c,\ a > 0, b \geq 0, c \geq 0\}$. Los estados muertos son omitidos.

Existen tres operaciones en los lenguajes regulares que permiten estudiar sus propiedades: *unión*, *concatenación* y *cerradura de Kleene*. Sean L, L_1 y L_2 lenguaje regulares:

- Unión: $L_1 \cup L_2 = \{w \mid w \in L_1 \ o \ w \in L_2\}$
- Concatenación: $L_1L_2 = \{wz \mid w \in L_1 \text{ y } \ z \in L_2\}$
- Cerradura de Kleene: $L^* = \cup_{i=0}^{\infty} L^i$, donde $L^0 = \{\varepsilon\}$, $L^1 = L$, $L^2 = LL$, $L^3 = LLL$, etc.

La cerradura de Kleene es una operación unaria que une cualquier cantidad de cadenas en L para formar un nuevo lenguaje. Por ejemplo, si $L_1 = \{10, 101\}$ y $L_2 = \{1, 01\}$, entonces:

$$L_1 \cup L_2 = \{1, 01, 10, 101\}$$

$$L_1L_2 = \{101, 1001, 1011, 10101\}$$

$$L_2^* = \{\varepsilon, 1, 01, 11, 011, 101, 0101, \ldots\}$$

Las operaciones regulares ayudan a construir *expresiones regulares* que definen lenguajes más complejos y las cuales describen el conjunto de todas las cadenas que se desean aceptar por el autómata finito.

La expresión $r \rightsquigarrow L$ se lee como "r es un expresión regular que denota al lenguaje L" y se define recursivamente por:

1. $\varnothing \rightsquigarrow \varnothing$,
2. $\varepsilon \rightsquigarrow \{\varepsilon\}$,
3. $a \rightsquigarrow \{a\}$, $\forall a \in \Sigma$,

   Ahora supóngase que $r_1 \rightsquigarrow L_1$ y $r_2 \rightsquigarrow L_2$, entonces:

4. $r_1 + r_2 \rightsquigarrow L_1 \cup L_2$,
5. $r_1 r_2 \rightsquigarrow L_1 L_2$,
6. $r_1^* \rightsquigarrow L_1^*$,
7. $(r_1) \rightsquigarrow L_1$.

Por ejemplo, si $\Sigma = \{0, 1\}$, entonces $0+1(0+1)^*$ es la expresión regular que denota a las cadenas binarias que representan los números naturales $\mathbb{N}$.

A pesar de que las expresiones regulares y los autómatas finitos describen de manera diferente a los lenguajes, cualquier expresión regular puede ser transformada en un autómata finito que reconozca el lenguaje que describe y viceversa, por lo tanto ambos reconocen a la clase de lenguajes regulares.

2.4. Lógica de primer orden

La *lógica de primer orden* (o *lógica de predicados*) permite el uso de *predicados*, los cuales son argumentos que contienen variables y cuantificadores de variables ($\forall$ y $\exists$). Por ejemplo, si $x, y \in \mathbb{Z}$, entonces el predicado $P(x, y) =$ "$x < y$", que contiene dos variables (x y y), será verdadero cuando x sea menor y, de lo contrario será falso. Las variables x y y representan a cualquier elemento de $\mathbb{Z}$, llamado (de manera general) *dominio* o *universo de discurso*, ya que define los objetos que son de interés en el argumento.

Un lenguaje de primer orden está formado por *términos* y *fórmulas*, los cuales son expresados por cadenas finitas de símbolos pertenecientes al alfabeto del lenguaje. Intuitivamente, un término representa un objeto del dominio de discurso y una fórmula expresa predicados que pueden ser verdaderos o falsos (cuando sus variables libres son interpretadas o instanciadas). El alfabeto se divide en símbolos lógicos (el símbolo de igualdad se considerará dentro de esta categoría) y símbolos no lógicos :

- Símbolos lógicos: Son aquellos que siempre tienen el mismo significado.
  1. Paréntesis, corchetes y otros símbolos de puntuación.
  2. Cuantificadores lógicos: $\forall$ (para todo) y $\exists$ (existe).
  3. Conectores lógicos: $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$.
  4. Variables (una por cada número entero positivo n): $v_1, v_2, v_3, \ldots$
  5. Símbolo de igualdad: $=$.
- Símbolos no lógicos: Son aquellos que su significado varía según la interpretación. El conjunto de todos los símbolos no lógicos es llamado *firma*.

1. Símbolos de predicado o relacionales: Para cada entero $n \geq 0$, se establece un conjunto de símbolos de predicado n-ario, los cuales representan relaciones entre n elementos.
2. Símbolos de función: Para cada entero positivo $n \geq 1$, se establece un conjunto de símbolos de función n-aria, los cuales reciben n parámetros formales.
3. Símbolos de constante: Es un conjunto, posiblemente vacío, de símbolos de funciones con aridad 0.

Los términos se definen como expresiones construidas a partir de símbolos constante y de variables mediante los símbolos de función. Entonces, cualquier variable y cualquier constante es un término y cualquier expresión $f(t_1, t_2, \ldots, t_n)$ con n argumentos (donde cada t_i es un término y f es un símbolo de función n-ario) es un término.

Las *fórmulas atómicas* son fórmulas que no contienen conectores ni cuantificadores lógicos. Las fórmulas se definen como expresiones construidas a partir de fórmulas atómicas y de conectores y cuantificadores lógicos. Formalmente se definen de la siguiente manera:

1. Si P es un símbolo de predicado n-ario y $t_1, t_2, \ldots, t_n$ son términos, entonces $P(t_1, t_2, \ldots, t_n)$ es una fórmula.
2. Si t_i y t_j son términos, entonces $t_i = t_j$ es una fórmula.
3. Si α y β son fórmulas, entonces $\neg\alpha$, $\alpha \wedge \beta$, $\alpha \vee \beta$, $\alpha \rightarrow \beta$ y $\alpha \leftrightarrow \beta$ son fórmulas.
4. Si α es una fórmula y x es una variable, entonces $\forall x\alpha$ y $\exists x\alpha$ son fórmulas.

Los paréntesis en cualquier fórmula sirven para garantizar que ésta se obtenga de una manera siguiendo la definición inductiva. En una fórmula, las variables pueden ser *libres* o *acotadas*. Estas últimas son aquellas que no están cuantificadas en la fórmula, por ejemplo, en la fórmula $\exists x P(x, y)$, y es la variable libre.

Existen convenciones que indican la precedencia de los operadores lógicos en una fórmula para omitir tantos paréntesis como sea posible y así obtener una fácil lectura de las mismas. Considérese que α y β son fórmulas, x es una variable, y t_1 y t_2 son términos:

1. Los paréntesis extremos se omiten. Entonces $\forall x\alpha \vee \beta$ es equivalente a $(\forall x\alpha \vee \beta)$.
2. Los operadores $\neg, \forall$ y $\exists$ se aplican primero. Por ejemplo:

$$\begin{array}{lll} \neg\alpha \wedge \beta & \text{es equivalente a } ((\neg\alpha) \wedge \beta), & \text{pero no a } \neg(\alpha \wedge \beta), \\ \forall x\alpha \vee \beta & \text{es equivalente a } ((\forall x\alpha) \vee \beta), & \text{pero no a } \forall x(\alpha \vee \beta), \\ \exists x\alpha \rightarrow \beta & \text{es equivalente a } ((\exists x\alpha) \rightarrow \beta), & \text{pero no a } \exists x(\alpha \rightarrow \beta). \end{array}$$

3. Los operadores $\wedge$ y $\vee$ se aplican después de la convención 2. Por ejemplo:

$$\neg\alpha \vee \beta \rightarrow \alpha \text{ es equivalente a } ((\neg\alpha) \vee \beta) \rightarrow \alpha.$$

4. Cuando un operador lógico se usa repetidamente, la fórmula se agrupa de izquierda a derecha. Por ejemplo:

$$\alpha \rightarrow \beta \rightarrow \neg\alpha \text{ es equivalente a } ((\alpha \rightarrow \beta) \rightarrow (\neg\alpha)).$$

Una *interpretación* I proporciona significado a los términos y predicados de un lenguaje; es decir, asigna a cada término un objeto del universo de discurso U y a cada predicado un valor (verdadero o falso). Entonces, la interpretación de un símbolo de predicado n-ario, es un conjunto de n-tuplas de elementos del universo de discurso, de tal manera que se puede establecer si el predicado es verdadero o no una vez que se especifiquen cuáles son los elementos del universo que están representados por sus variables libres. Mientras que las interpretaciones de un símbolo de función n-ario y de un símbolo de constante, son respectivamente, una función $f : U^n \rightarrow U$ y una función de aridad 0 (a la que simplemente se le considera como un elemento de U). Por ejemplo, si $U = \mathbb{Z}$, f es un símbolo de función binario y c un símbolo de constante, entonces $I(f)$ puede significar la multiplicación de dos argumentos y $I(c) = 25$ (es decir, a c se le asigna el valor 25).

Una interpretación aplicada a los elementos de un lenguaje formal define una *estructura*. Dada una firma σ, una σ-estructura (o *modelo*) es una tupla $\mathfrak{A} = \langle A, \{c_i^{\mathfrak{A}}\}, \{P_i^{\mathfrak{A}}\}, \{f_i^{\mathfrak{A}}\}\rangle$ que consiste de un conjunto no vacío A que forma el *dominio de discurso* (o universo de discurso) y de una interpretación I de σ, donde:

1. A cada símbolo de predicado n-ario $P_i^{\mathfrak{A}}$ le es asignado una relación $I(P_i^{\mathfrak{A}})$ sobre A^n, es decir, un subconjunto de A^n o equivalentemente una función de A^n a $\{falso, verdadero\}$.

2. A cada símbolo de constante c_i le es asignado un elemento de $c_i^{\mathfrak{A}}$ de A.

3. A cada símbolo de función n-aria f_i le es asignado una función $f_i^{\mathfrak{A}} : A^n \rightarrow A$.

Si la estructura $\mathfrak{A}$ tiene a un conjunto finito como universo de discurso, entonces se dice que la estructura $\mathfrak{A}$ es finita.

Un ejemplo de una estructura es $\mathfrak{A} = \langle \mathbb{N}, <^{\mathfrak{A}} \rangle$, donde $<^{\mathfrak{A}}$ establece el conjunto de parejas (m, n) tal que $m < n$. Entonces, la fórmula $P^{\mathfrak{A}}(x, y) = \exists x \forall y \neg y < x$, que es traducida en $\mathfrak{A}$ como "existe un número natural tal que no hay número natural menor que él", es verdadera.

Una *asignación de variables* μ es una función que asocia a cada término con un elemento del universo de discurso; en particular, da significado a las fórmulas con variables libres. Si I es una interpretación, entonces la asignación se realiza de la siguiente manera:

1. Cada variable x evalúa a $\mu(x)$.

2. Dada un símbolo de función n-ria f y los términos $t_1, t_2, \ldots, t_n$, entonces

$$\mu(f(t_1, t_2, \ldots, t_n)) = (I(f))(\mu(t_1), \mu(t_2), \ldots, \mu(t_n)).$$

Ahora a cada fórmula se le asigna un valor de verdad a través de la siguiente definición inductiva:

1. Fórmulas atómicas: Si $P(t_1, t_2, \ldots, t_n)$ es una fórmula, $I(P)$ una interpretación de P y $\mu(t_1) = d_1, \mu(t_2) = d_2, \ldots, \mu(t_n) = d_n$, entonces a P se le asocia el valor verdadero o falso dependiendo de si $(d_1, d_2, \ldots, d_n) \in I(P)$, el cual es un subconjunto de A^n. Por otra parte, la fórmula $t_1 = t_2$ es verdadera si $\mu(t_1) = \mu(t_2)$.

2. Fórmulas con conectivos lógicos: Las fórmulas $\neg\alpha$, $\alpha \wedge \beta$, $\alpha \vee \beta$, $\alpha \rightarrow \beta$ y $\alpha \leftrightarrow \beta$ se evalúan a través de las tablas de verdad de los conectivos lógicos, como en la lógica proposicional.

3. Fórmulas con cuantificadores lógicos: La fórmula $\forall x \alpha(x)$ es verdadera si para cada valor de la variable x, $(I(\alpha))(\mu(x))$ es verdadera. Pero si la fórmula es $\exists x \alpha(x)$, entonces ésta es verdadera si para un valor de la variable x, $(I(\alpha))(\mu(x))$ es verdadera.

Dada una fórmula α (o un conjunto de fórmulas φ), un universo A y una interpretación I, se dice que I *satisface* a α (o a φ), denotado respectivamente por $I \vDash \alpha$ o $I \vDash \varphi$, si α (o cada una de las fórmulas en φ) es verdadera bajo la interpretación I. Cuando las fórmulas contienen variables libres, por convención se dice que una interpretación las satisface si la fórmula es verdadera independientemente de los valores del dominio de discurso que tomen las variables libres.

Se dice que un fórmula α es *lógicamente válida* si es verdadera en toda interpretación (como las tautologías en la lógica proposicional) bajo cualquier universo. Mientras que α es *consecuencia lógica* de una fórmula β, para cualquier universo, si toda interpretación que satisface a β también satisface a α.

Una *teoría de primer orden* sobre una firma σ es un conjunto de sentencias. Se dice que una σ-estructura $\mathfrak{A}$ es un *modelo* de una teoría T si y sólo si para cada sentencia φ de T, la estructura $\mathfrak{A}$ es un modelo de φ, $\mathfrak{A} \vDash \varphi$. Además, la teoría T es llamada *consistente* si tiene un modelo. Mientras que una *clase elemental* es el conjunto de todas las estructuras que satisfacen una teoría T.

Por ejemplo, la firma σ de la teoría de primer orden de gráficas T_G sólo tiene una relación binaria R, $R(x, y)$, que indica la adyacencia entre el vértice x y el vértice y ($x \sim y$). Entonces, la σ-estructura $\mathfrak{A}$ está formada por la tupla $\langle V, R \rangle$, donde V es el dominio de discurso formado por vértices. Las sentencias (axiomas) en T_G son las siguientes:

- $\forall x \forall y R(x,y) \rightarrow R(y,x)$, indica que las aristas son no dirigidas.
- $\forall x \neg R(x,x)$, indica que ningún vértice tiene lazos.

2.5. Estructuras automáticas

En lo que sigue se considerará el símbolo especial $\diamond \notin \Sigma$ que denotará un relleno en la parte derecha de las cadenas. Sea Σ un alfabeto, entonces el alfabeto extendido $\Sigma \cup \{\diamond\}$ será denotado por $\Sigma_\diamond$. Si $w \in \Sigma$ se define $w[i] = \diamond$ para toda $i > |w|$. Para cualquier conjunto A el *producto cartesiano* de n copias de A es denotado por A^n.

Dada una secuencia de cadenas $\overline{w} = (w_1, w_2, \ldots, w_n)$ sobre Σ, su *convolución* $u = \otimes\overline{w} = \otimes(w_1, w_2, \ldots, w_n)$ es la cadena de longitud $|u| = \max|w_i|$ sobre el alfabeto $(\Sigma_\diamond)^n$ cuyo i-ésimo símbolo es $u[i] = (w_1[i], w_2[i], \ldots, w_n[i])$. Por ejemplo: $\otimes(101, 11011) = (1,1)(0,1)(1,0)(\diamond,1)(\diamond,1)$ es la cadena de longitud 5 sobre el alfabeto:

$$(\Sigma_\diamond)^2 = \{(0,0),(0,1),(0,\diamond),(1,0),(1,1),(1,\diamond),(\diamond,0),(\diamond,1),(\diamond,\diamond)\}.$$

Un *autómata finito síncrono* con n entradas es un autómata que lee simultáneamente un número finito n de cadenas de entrada. Este autómata es equivalente al AFD sobre un alfabeto diferente, es decir, sobre el alfabeto $(\Sigma_\diamond)^n$. Por ejemplo, sea M un autómata finito síncrono con 3 entradas sobre el alfabeto $\Sigma = \{0,1\}$, el cual acepta la tripleta de cadenas (w_1, w_2, w_3), donde $w_1 = 011$, $w_2 = 1000$, $w_3 = 0000$; entonces M es equivalente a un AFD sobre el alfabeto $(\Sigma_\diamond)^3$ quien acepta a $\otimes(w_1, w_2, w_3) = (0,1,0)(1,0,0)(1,0,0)(\diamond,0,0)$, como se puede ver en la figura 2.8.

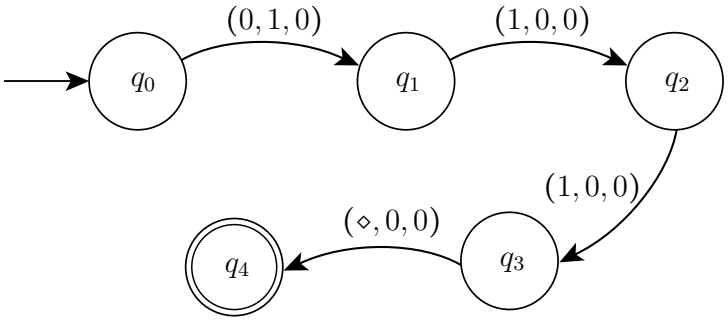


Figura 2.8: Diagrama del AFD que acepta a la cadena formada por la convolución de $(011, 1000, 0000) \in (\Sigma_\diamond)^3$. Los estados muertos son omitidos.

Nótese que la convolución $\otimes(w_1, w_2, w_3)$ agrupó tres símbolos en cada paréntesis, lo que asemeja a las tres cintas del autómata finito síncrono leyendo simultáneamente a los tres símbolos correspondientes de la tripleta (w_1, w_2, w_3). De esta manera no se necesita ningún modelo nuevo de computación, y además, toda la teoría en AFD puede ser fácilmente aplicada a los autómatas finitos síncronos. Por esta razón,

de aquí en adelante, cualquiera de estos dos tipos de autómatas que se describan simplemente serán llamados autómatas.

Una *estructura relacional* es una tupla $\mathcal{A} = (A, R_1, \ldots, R_n)$ donde A es un conjunto numerable (finito o infinito), llamado *dominio* y R_i son las relaciones en A con alguna aridad $r_i \in \mathbb{N}$, es decir, $R_i \subseteq A^{r_i}$. Si el dominio es el conjunto de cadenas de Σ^*, se puede definir la convolución de una relación $R \subseteq (\Sigma^*)^r$ como el conjunto $\otimes R = \{\otimes\overline{w} : \overline{w} \in R\}$, es decir, $\otimes R \subseteq ((\Sigma_\diamond)^r)^*$ es el conjunto de convoluciones de las tuplas en R. Obsérvese que $\otimes R$ es un lenguaje sobre el alfabeto $(\Sigma_\diamond)^r$ y que si existe un autómata M que lo reconozca, $L(M) = \otimes R$, entonces $\otimes R$ es un lenguaje regular. Además, si $L(M) = \otimes R$, entonces R es una *relación automática*.

Por ejemplo, considérese la relación $R_S \in \mathtt{N}$, donde $R_S = \{(t, t+1) \mid t \in \mathtt{N}\}$ es la relación del sucesor de t en el conjunto de las cadenas binarias que representan a los números naturales (ver sección 2.3). Entonces, R_S es automática ya que su convolución es reconocida por el autómata que se muestra en la figura 2.9 y denotada por la siguiente expresión regular (sobre el alfabeto $(\Sigma_\diamond)^2$):

$$(0,1) + (1,1)(1,0)^*(\diamond,0) + (1,1)((1,1) + (0,0))^*(0,1)(1,0)^*$$

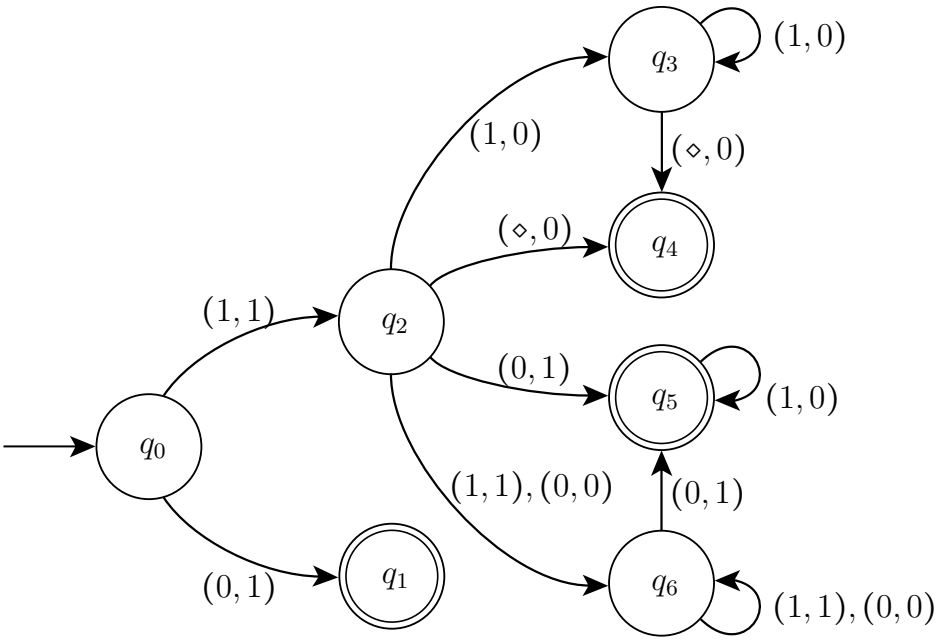


Figura 2.9: Diagrama del AFD que acepta al lenguaje regular generado por la convolución de las cadenas de R_S. Los estados muertos son omitidos.

Entonces, si los pares ordenados $(0,1)$, $(111,1000)$ y $(1100,1101) \in R_S$, la convolución respectiva a cada una es $(0,1)$, $(1,1)(1,0)(1,0)(\diamond,0)$ y $(1,1)(1,1)(0,0)(0,1)$,

las cuales son cadenas que están en $\otimes R_S$ y son generadas por la expresión regular anterior y reconocidas por el autómata anterior.

Las relaciones automáticas tienen propiedades muy fuertes de decibilidad y de cerradura, las cuales son heredadas de la teoría de autómatas: cada relación finita es automática, además son cerradas bajo proyección, instanciación, cilindrificación (ver figura 2.10), permutación de coordenadas (ver figura 2.11) y muchas otras [54]. En particular, las relaciones automáticas son cerradas bajo construcciones lógicas:

Teorema 2.5.1 (Khoussainov y Nerode. Teorema 4.4 en [30]). *Si R_1, R_2 son relaciones automáticas, entonces $R_1 \vee R_2$, $R_1 \wedge R_2$, $\neg R_1$, $\exists x(R_1)$ y $\forall x(R_1)$ también son relaciones automáticas. Además, el autómata para esas relaciones puede ser algorítmicamente construido del autómata de R_1 y R_2.*

El autómata que reconoce a las operaciones $R_1 \vee R_2$ y $R_1 \wedge R_2$ se obtiene, respectivamente, por la unión e intersección de los autómatas de R_1 y R_2 (nótese que si la relación R_1 tiene una aridad menor que la de R_2, primero se tiene que cilindrificar R_1 agregando las coordenadas necesarias para que ambas relaciones automáticas tengan la misma aridad). Mientras que el autómata que reconoce a $\neg R_1$ se genera por medio del complemento del autómata de R_1.

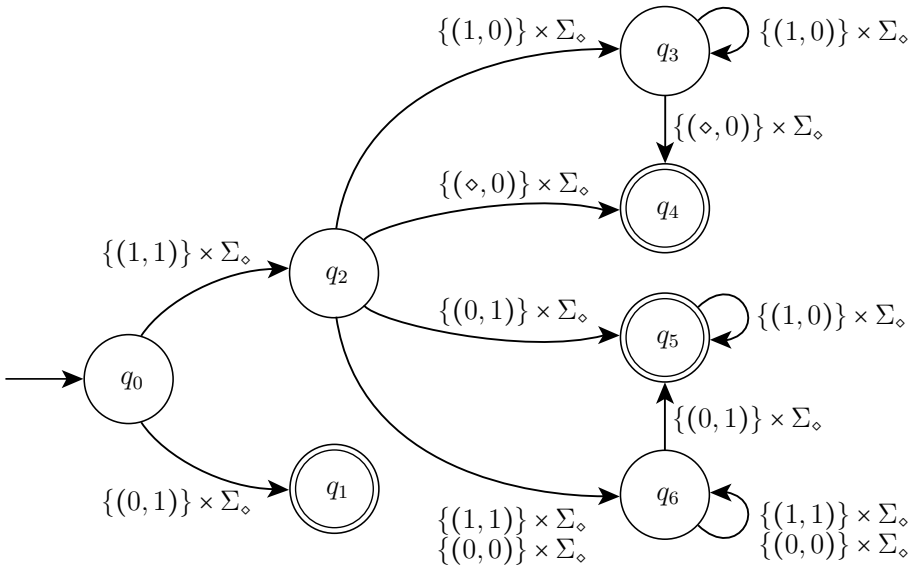


Figura 2.10: Diagrama del AFD que acepta al lenguaje regular generado por la convolución de las cadenas de $R'_S = \{(t, t+1, z) \mid t \in \mathbb{N} \text{ y } z \in \Sigma_\diamond\}$, la cual establece la cilindrificación de la relación automática R_S al agregar una tercera coordenada. De manera análoga se procede a cilindrificar a R_S si se agrega dos o más coordenadas. Los estados muertos son omitidos.

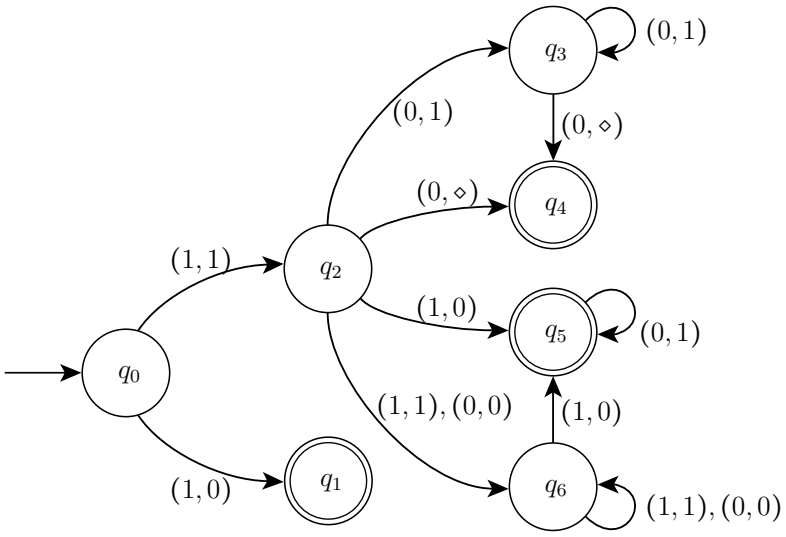


Figura 2.11: Diagrama del AFD que acepta al lenguaje regular generado por la convolución de las cadenas de $R''_S = \{(t+1, t) \mid t \in \mathbb{N}\}$, la cual establece la permutación de coordenadas de la relación automática R_S. Los estados muertos son omitidos.

Suponiendo que R_1 tiene aridad n, las relaciones automáticas $\exists x(R_1)$ y $\forall x(R_1)$ son definidas como:

$$\exists x R_1 = \{(x_1, \ldots, x_{i-1}, x_i, \ldots, x_n) \mid \exists x \in (\Sigma_\diamond)^*(x_1, \ldots, x_{i-1}, x, x_i, \ldots, x_n) \in R_1\}$$

$$\forall x R_1 = \{(x_1, \ldots, x_{i-1}, x_i, \ldots, x_n) \mid \forall x \in (\Sigma_\diamond)^* \ (x_1, \ldots, x_{i-1}, x, x_i, \ldots, x_n) \in R_1\}.$$

Entonces, si $M_1 = (Q_1, (\Sigma_\diamond)^n, \delta_1, q_0^1, F_1)$ es el autómata que reconoce a la relación R_1, el autómata finito determinista que reconoce a $\exists x(R_1)$ es $M'_1 = (Q'_1, (\Sigma_\diamond)^n, \delta'_1, q_0^{1'}, F'_1)$ donde:

1. $Q'_1 = \{2^{Q_1}\}$,
2. $q_0^{1'} = \{q_0^1\}$,
3. $\delta'_1(\mathrm{X}, a) = \bigcup_{\substack{q \in \mathrm{X} \\ x_i \in (\Sigma_\diamond)^*}} \{\delta_1(q, (x_1, \ldots, x_{i-1}, a, x_{i+1}, \ldots, x_n))\}$ y
4. $F'_1 = \{\mathrm{X} \in Q'_1 \mid \mathrm{X} \bigcap F_1 \neq \varnothing\}$.

El autómata finito determinista que reconoce a la relación $\forall x(R_1)$ se define igual al anterior, excepto que el conjunto de estados finales es $\{\mathrm{X} \in Q'_1 \mid \mathrm{X} \subseteq F_1\}$.

Cuando R_1 es una relación unaria, ambos $\exists x(R_1)$ y $\forall x(R_1)$ son relaciones 0-arias las cuales, estrictamente hablando, no pueden ser automáticas por razones técnicas,

pero en [30] usan relaciones anurias ($R = (\Sigma_\diamond)^*$ para la relación `verdadera` y $R = \varnothing$ para la relación `falsa`) para *representar* a esas relaciones 0-arias.

Para dar una presentación de la estructura relacional $\mathcal{A} = (A, R_1, \ldots, R_n)$ por medio de autómatas, se necesita representar cada elemento de A por una o más cadenas del conjunto L, donde $L \subseteq \Sigma^*$ para algún alfabeto Σ y L un lenguaje regular, entonces $L = L(M_A)$ para un autómata M_A. Un mapeo $\mu : L \to A$ indica que un elemento de $a \in A$ es representado por alguna cadena $w \in L$, cuando $\mu(w) = a$. También las relaciones $R_i \subseteq A^{r_i}$ serán representadas por las relaciones correspondientes $\mu^{-1}(R_i) \subseteq L^{r_i}$ definidas por $\mu^{-1}(R_i) = \{(w_1, w_2, ..., w_{r_i}) \in L^{r_i} \mid (\mu(w_1), \mu(w_2), \ldots, \mu(w_{r_i})) \in R_i\}$.

Una *presentación automática* de una estructura relacional $\mathcal{A} = (A, R_1, \ldots, R_n)$ es un mapeo μ y una n-tupla de autómatas $(M_A, M_1, \ldots, M_n)$ tal que:

1. $\mu : L(M_A) \to A$ es suprayectiva.
2. $L(M_i) = \otimes\mu^{-1}(R_i) \subseteq ((\Sigma_\diamond)^{r_i})^*$.

El siguiente teorema menciona una de muchas propiedades interesantes de decibilidad que tienen las estructuras automáticas:

Teorema 2.5.2 (Khoussainov y Nerode. Corolario 4.2 en [30])**.** *La teoría de primer orden de cualquier estructura automática es algorítmicamente decidible.*

Además, el teorema 2.5.2 ha sido generalizado más allá de la lógica de primer orden para incluir varios cuantificadores generalizados (incluyendo $\exists^\omega$, $\exists^{(k,m)}$ y $\exists^{k-ram}$) e incluso para algunos fragmentos de la lógica de segundo orden llamada FSO.

Usualmente, la relación de identidad en A está incluida entre las relaciones R_i y por lo tanto, como parte de las presentación automática, un autómata $M_=$ es dado, el cual es capaz de reconocer dos representaciones diferentes w_1, w_2 por el mismo elemento $a \in A$, es decir: $L(M_=) = \{\otimes(w_1, w_2) : \mu(w_1) = \mu(w_2)\}$. Aquí, se optará usar el enfoque equivalente en el cual μ es inyectiva y por lo tanto $M_=$ es un autómata muy simple ($L(M_=) = \{\otimes(w, w) : w \in L\}$) y los elementos de A pueden ser identificados con su representación en $L = L(M_A)$. Entonces, se omitirá el autómata $M_=$ en lo sucesivo.

Una *gráfica automática* es un par ordenado de autómatas $G = (M_V, M_E)$, donde los alfabetos de M_V y M_E son respectivamente Σ y $(\Sigma_\diamond)^2$. Los vértices y aristas de G están dados por $V = L(M_V)$ y $\otimes E = L(M_E)$. Nótese que una gráfica automática es, entonces, una presentación automática de la gráfica $(V(G), V(E))$ con $\mu = 1_{V(G)}$, pero se preferirá considerar que una gráfica automática es una gráfica por derecho propio. Se denotará por $\mathcal{AG}$ a la clase de todas las gráficas automáticas.

Por ejemplo, sea $G \in \mathcal{AG}$ la gráfica que se muestra en la figura 2.12 (a). La gráfica automática correspondiente a G está dada por $G_A = (M_V, M_E)$, donde $\Sigma = \{0, 1, \ldots, 8\}$ es el alfabeto del autómata M_V (ver figura 2.12 (b)) que reconoce a los vértices de G, es decir, $V = L(M_V)$. Además, el autómata M_E (ver 2.12 (c)) reconoce a la convolución de todas las aristas de G, de tal manera que si $uv \in G$, entonces M_E reconoce a $\otimes(u, v)$ y a $\otimes(v, u)$.

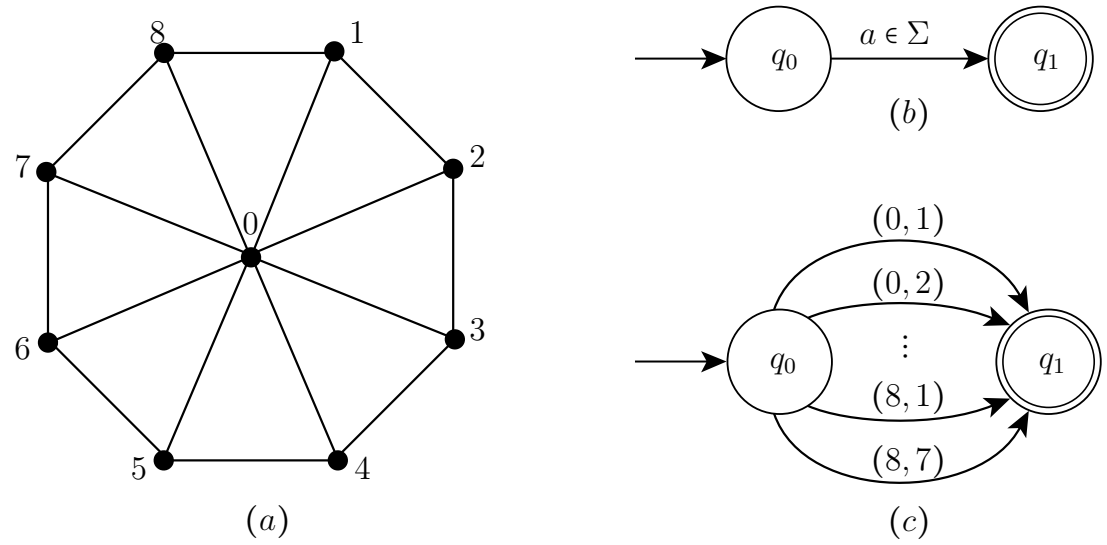


Figura 2.12: (a) Gráfica G. (b) y (c) se muestran los diagramas de los autómatas M_V y M_E que reconocen a $V(G)$ y $\otimes E(G)$, respectivamente; donde $\Sigma = \{0, 1, \ldots, 8\}$ (en ambos casos, los estados muertos son omitidos). Las flechas del autómata M_E están ordenadas de manera lexicográfica de arriba hacia abajo.

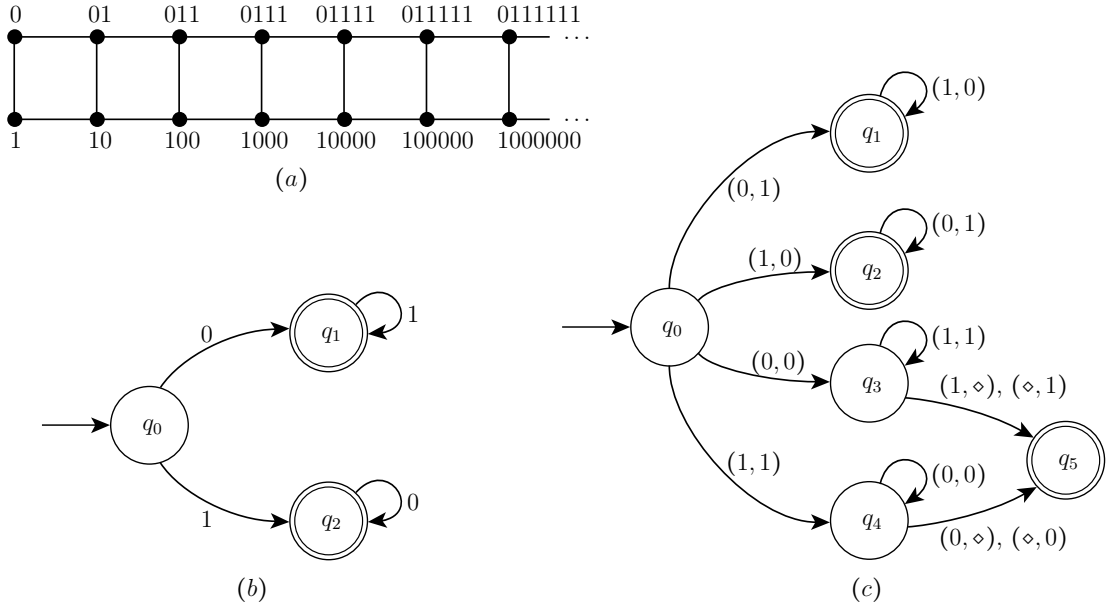


Figura 2.13: (a) Gráfica infinita G'. Mientras que en (b) y (c) se muestran los diagramas de los autómatas que reconocen a los vértices y aristas de G', respectivamente (en ambos, los estados muertos son omitidos).

Considérese una gráfica infinita G' tal que sus conjuntos de vértices y aristas son infinitos. La gráfica automática correspondiente a G' puede obtenerse si la etiquetación de los vértices de G' permite establecer un patrón que reconozca todas sus aristas por medio de un autómata. Por ejemplo, si G' es la gráfica que se muestra en la figura 2.13 (a), los autómatas que conforman a su gráfica automática son los que aparecen en la figura 2.13 (b) y (c), los cuales respectivamente reconocen a los vértices y aristas de G'.

2.6. Máquinas de Turing

La *máquina de Turing (MT)*, propuesta por Alan Turing en 1936, es un modelo teórico de una computadora capaz de realizar todo lo que una computadora actual puede hacer. La MT cuenta con una *unidad de control* (UC), con una *cabeza* y con una cinta infinita (ver figura 2.14). La cabeza lee y escribe símbolos en la cinta, y también realiza movimientos a la izquierda y a la derecha a través de la misma. Los movimientos están en función del estado de la unidad de control y del símbolo que la cabeza está apuntando en la cinta. Inicialmente en la cinta se encuentra una cadena de entrada w y en el resto, el símbolo correspondiente del espacio en blanco de la MT ("␣"). Se dice que la MT *acepta* a w, si la máquina de Turing pasa en algún momento por algún estado de aceptación; en caso contrario w es *rechazada*.

Una posibilidad es que la máquina de Turing nunca se detenga, pero aún así, la cadena w se considera aceptada o rechazada, dependiendo de si la máquina de Turing paso no por algún estado de aceptación en algún momento.

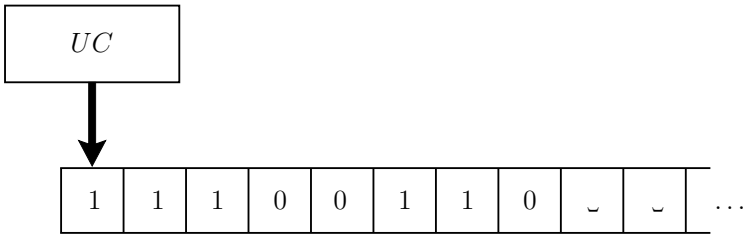


Figura 2.14: Esquema de una máquina de Turing. La cabeza es representada mediante la flecha que está entre la unidad de control y la cinta.

Una máquina de Turing es una 7-tupla, $(Q, \Sigma, \Gamma, \delta, q_0, \text{␣}, F)$, donde:

1. Q es el conjunto (finito) de estados,
2. Σ es el alfabeto (finito) de entrada que no contiene el símbolo correspondiente al espacio en blanco,
3. Γ es el alfabeto (finito) de la cinta el cual contiene a los símbolos del conjunto $\{\Sigma \cup \text{␣}\}$.

4. $\delta:\ Q \times \Gamma \longrightarrow Q \times \Gamma \times \{\leftarrow, \rightarrow\}$ es la función (parcial) de transición,

5. $q_0 \in Q$ es el estado inicial,

6. $\llcorner \in \{\Gamma \setminus \Sigma\}$ es el espacio en blanco en la cinta,

7. F, tal que $F \subseteq Q$, es el conjunto de estados de aceptación.

La ejecución de una máquina de Turing dependerá de la función de transición δ, ya que es la que contiene las reglas de los movimientos que realizará dependiendo del símbolo que la cabeza lea en la cinta y del estado en el que se encuentre; entonces, δ indicará qué símbolo escribirá en la cinta, a qué estado cambiará y la dirección hacia donde se tiene que mover la cabeza (izquierda $\leftarrow$ o derecha $\rightarrow$). Para representar estos movimientos se utilizan las *configuraciones* o *descripciones instantáneas* de una máquina de Turing, las cuales son cadenas del tipo $s = uqv \in \Gamma^* Q \Gamma^*$, donde q es el estado actual, uv es la cadena actual que está en la cinta y la cabeza está posicionada en el primer símbolo de la cadena v. Por ejemplo, $1110q_41010$ representa la configuración cuando en la cinta está la cadena $w = 11101010$, el estado actual es q_4 y la cabeza está posicionada en el quinto símbolo de w, es decir, el primer símbolo de v.

Para indicar que una configuración s transitó en un sólo paso a otra configuración s', se escribe $s \vdash s'$. Por ejemplo, $1110q_41010 \vdash 11101q_3010$ si la función de transición indica que $\delta(q_4, 1) = (q_3, 1, \rightarrow)$, es decir, se pasó al estado q_3, escribió 1 en la cinta y se movió a la derecha. Cuando una máquina de Turing pasa de una configuración a otra en un número de pasos igual o mayor a cero, se denota $s \overset{*}{\vdash} s'$.

Una máquina de Turing M reconoce a un lenguaje L si M acepta a todas las cadenas de L y a ninguna otra; lo cual se denota por $L = L(M)$. Un lenguaje que es reconocido por una máquina de Turing es llamado *recursivamente enumerable*. Si además, el lenguaje es reconocido por alguna máquina de Turing que siempre para (sin importar la cadena de entrada), entonces, el lenguaje es *recursivo*.

Ejemplo 2.6.1. *La máquina de Turing que reconoce al lenguaje de las cadenas binarias que representan a los números naturales (ver ejemplo 2.3.2), está dada por la tupla* $(Q, \Sigma, \Gamma, \delta, q_0, \llcorner, F)$*, donde:*

1. $Q = \{q_0, q_1, q_2, q_3, q_4\}$,

2. $\Sigma = \{0, 1\}$,

3. $\Gamma = \{0, 1, \llcorner\}$,

4. $\delta:$

	0	1	$\sqcup$
q_0	–	$(q_1, 1, \rightarrow)$	–
q_1	$(q_2, 0, \rightarrow)$	$(q_1, 1, \rightarrow)$	–
q_2	$(q_3, 0, \rightarrow)$	$(q_2, 1, \rightarrow)$	$(q_4, \sqcup, \leftarrow)$
q_3	$(q_3, 0, \rightarrow)$	–	$(q_4, \sqcup, \leftarrow)$
q_4	–	–	–

5. q_0 *es el estado inicial,*

6. $F = \{q_4\}$.

Dada la máquina de Turing anterior, las configuraciones que se generarán con la entrada $w = 11011$ son las siguientes:

$$q_0 11011 \vdash 1q_1 1011 \vdash 11q_1 011 \vdash 110q_2 11 \vdash 1101q_2 1 \vdash 11011q_2 \vdash 1101q_4 1$$

Dado que la sucesión de descripciones instantáneas terminan en el estado q_4 y éste es un estado final, la máquina de Turing se detendrá y aceptará a w.

Nótese, que dadas dos configuraciones s y s' de una máquina de Turing M, la notación $s \vdash s'$ representa una relación binaria entre ambas configuraciones, por lo que esta relación puede ser considerada como una relación de adyacencia (asimétrica) sobre el conjunto de todas las configuraciones.

Observación 2.6.2. *Si D es el conjunto de todas las configuraciones de M, entonces D se puede considerar como una digráfica con la relación de adyacencia $\vdash$.*

En una transición $s \vdash s'$ de una máquina de Turing M, sólo los símbolos que están a la izquierda y a la derecha del estado son los que cambian en las cadenas que representan a s y s' independientemente del movimiento que M realice en la transición. Por ejemplo, considérese la siguiente máquina de Turing:

Ejemplo 2.6.3. *Sea M_P una máquina de Turing sobre el alfabeto $\Sigma_P = \{a, b\}$ que reconoce cadenas que tienen un número par de ocurrencias del símbolo* a*, la cual está dada por la tupla $(Q_P, \Sigma_P, \Gamma_P, \delta, p_0, \sqcup, F_P)$, donde:*

1. $Q_P = \{p_0, p_1, p_2\}$,

2. $\Sigma_P = \{a, b\}$,

3. $\Gamma_P = \{a, b, \sqcup\}$,

4. δ_P :

	a	b	$\sqcup$
p_0	$(p_1, a, \rightarrow)$	$(p_0, b, \rightarrow)$	$(p_2, \sqcup, \leftarrow)$
p_1	$(p_0, a, \rightarrow)$	$(p_1, b, \rightarrow)$	–
p_2	–	–	–

5. p_0 *es el estado inicial,*

6. $F_P = \{p_2\}$.

Si la cadena de entrada a M_P es $w = ababb$ entonces se tendrán las siguientes transiciones:

$$p_0ababb \vdash ap_1babb \vdash abp_1abb \vdash abap_0bb \vdash ababp_0b \vdash ababbp_0 \vdash ababp_2b\textvisiblespace.$$

Si las configuraciones involucradas en cada transición se escriben una debajo de otra resulta fácil observar la longitud de las cadenas que las representan y los símbolos que las hacen diferentes. Por ejemplo, sin pérdida de generalidad, considérese a las transiciones $abp_1abb \vdash abap_0bb$ y $ababbp_0 \vdash ababp_2b\textvisiblespace$ en las cuales M_P se mueve hacia la derecha e izquierda, respectivamente. En la tabla 2.1 se observa que las configuraciones que conforman a la transición $abp_1abb \vdash abap_0bb$ tienen la misma longitud y que los únicos símbolos que las hacen distintas son los que están en las columnas de las posiciones 3 y 4; mientras que en la tabla 2.2 se observa que las configuraciones de la transición $ababbp_0 \vdash ababp_2b\textvisiblespace$ tienen una longitud distinta (ya que M_P escribe el símbolo $\textvisiblespace$ y posteriormente se mueve a la izquierda) y que sus símbolos difieren sólo en las posiciones $5, 6$ y 7. En la posición 7 de la tabla 2.2 podría existir un símbolo distinto, ya que otra máquina de Turing podría escribir cualquier símbolo distinto de "$\textvisiblespace$" y posteriormente moverse hacia la izquierda (segundo renglón).

Posición	1	2	3	4	5	6
	a	b	p_1	a	b	b
	a	b	a	p_0	b	b

Tabla 2.1: Las cadenas de los renglones 1 y 2 corresponden respectivamente a las configuraciones abp_1abb y $abap_0bb$. Ambas configuraciones pertenecen a la transición $abp_1abb \vdash abap_0bb$, donde M_P se mueve a la derecha. Sólo los símbolos son distintos en las columnas de las posiciones 3 y 4.

Posición	1	2	3	4	5	6	7
	a	b	a	b	b	p_0	
	a	b	a	b	p_2	b	$\textvisiblespace$

Tabla 2.2: Las cadenas de los renglones 1 y 2 corresponden respectivamente a las configuraciones $ababbp_0$ y $ababp_2b\textvisiblespace$. Ambas configuraciones pertenecen a la transición $ababbp_0 \vdash ababp_2b\textvisiblespace$, donde M_P se mueve a la izquierda. Sólo los símbolos son distintos en las columnas de las posiciones $5, 6$ y 7.

Por otra parte, nótese que el i-ésimo par ordenado de la convolución $\otimes$ = $(abp_1abb, abap_0bb)$ = $(a,a)(b,b)(p_1,a)(a,p_0)(b,b)(b,b)$ corresponde a los símbolos que se encuentran en la i-ésima posición (leyendo de arriba hacia abajo) de la tabla 2.1; y que lo mismo sucede con la tabla 2.2 y la convolución $\otimes$ = $(ababbp_0, ababp_2b_\sqcup)$ = $(a,a)(b,b)(a,a)(b,b)(b,p_2)(p_0,b)(\diamond,\sqcup)$ a excepción de que en la posición 7 no existe el símbolo $\diamond$ en la primera fila. Esto indica que ambas transiciones pueden ser descritas por medio de su convolución sobre el alfabeto $(\Gamma_P \cup Q_P)^2_\diamond$.

Por lo tanto, los pequeños cambios de símbolos que hay de una configuración a otra, dada cualquier transición de M_P, son fácilmente reconocidos por un autómata finito determinista M'_P sobre el alfabeto $(\Gamma_P \cup Q_P)^2_\diamond$, donde M'_P acepta la convolución de cualesquiera dos configuraciones s y s', tal que $s \vdash s'$. El autómata M'_P debe de revisar dos o tres posibles cambios en las cadenas que aceptará, si respectivamente s y s' tienen igual o diferente longitud; además debe de verificar que no haya más cambios que esos.

En la figura 2.15 se muestra el diagrama de estados de M'_P que reconoce la relación de transición de la máquina de Turing M_P del ejemplo 2.6.3.

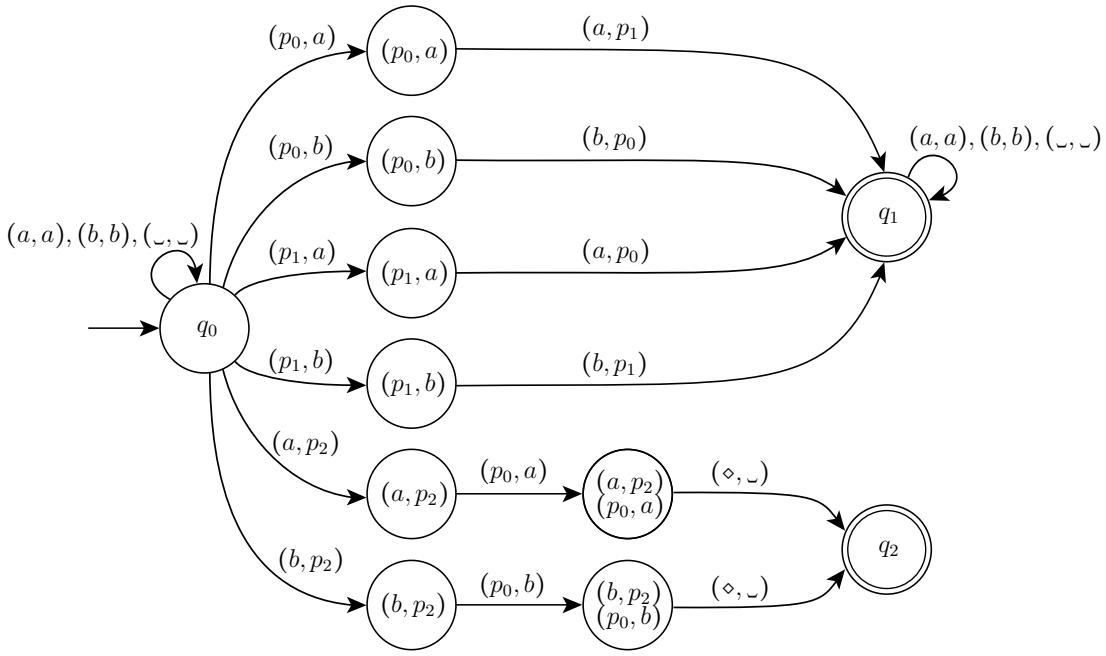


Figura 2.15: Diagrama del AFD que reconoce a las transiciones $s \vdash s'$ de la máquina de Turing M_P (ver ejemplo 2.6.3) por medio de las convoluciones $\otimes(s,s')$. Se llegará a algún estado final (q_1 o q_2) sólo cuando $s \vdash s'$; de lo contrario se alcanzará al estado muerto, el cual es omitido en el diagrama.

El diagrama de estados del autómata genérico que reconoce a todas las transiciones de cualquier máquina de Turing $M = (Q, \Sigma, \Gamma, \delta, p_0, \sqcup, F)$ se muestra en la figura

2.16. En éste se omiten los estados muertos.

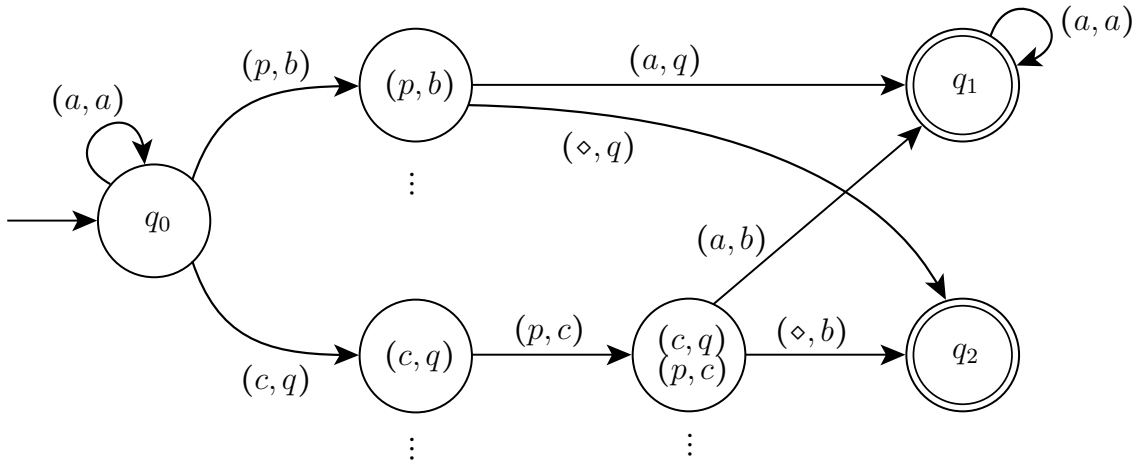


Figura 2.16: Diagrama del AFD genérico que reconoce a las transiciones $s_i \vdash s_j$ que se generan con una máquina de Turing $M = (Q, \Sigma, \Gamma, \delta, p_0, \llcorner, F)$ por medio de las convoluciones $\otimes(s_i, s_j)$. Los símbolos a, b y $c \in \Gamma$; mientras que p y $q \in Q$. Los estados muertos son omitidos en el dibujo.

El siguiente teorema es enunciado en las fuentes [4] y [30] pero no es probado explicítamente en ninguna de ellas. Para comodidad del lector, se extiende la prueba completa a continuación:

Teorema 2.6.4 (Khoussainov y Nerode. Prop. 2.6 en [30], Blumensath y Grädel. Lema 5.12 en [4])**.** *Para toda máquina de Turing, la relación de transición* $uqv \vdash u'q'v'$ *es automática.*

Demostración. Sea $M = (Q, \Sigma, \Gamma, \delta, p_0, \llcorner, F)$ una máquina de Turing. Entonces, el autómata $M' = (Q', \Sigma', \delta', q_0, F')$ que reconoce todas las transiciones de M está dado como sigue (recuérdese que † representa el estado muerto del AFD):

1. $Q' = \{q_0, q_1, q_2\} \cup (Q \times \Gamma) \cup (\Gamma \times Q) \cup [(\Gamma \times Q) \times (Q \times \Gamma)] \cup \{\dagger\}$

2. $\Sigma' = (\Gamma \cup Q)^2_\diamond \backslash \{(\llcorner, \diamond), (\diamond, \diamond)\}$,

3. q_0 es el estado inicial,

4. δ' :
   $\delta'(q_0,(a,a)) = q_0,\ \forall a \in \Gamma$
   $\delta'(q_0,(p,b)) = (p,b),\ \forall p \in Q$ y $b \in \Gamma$
   $\delta'(q_0,(c,q)) = (c,q),\ \forall q \in Q$ y $c \in \Gamma$
   $\delta'((p,b),(a,q)) = q_1$, si $\delta(p,a) = (q,b,\rightarrow)$
   $\delta'((p,b),(\diamond,q)) = q_2$, si $\delta(p,\llcorner) = (q,b,\rightarrow)$
   $\delta'((c,p),(p,c)) = ((c,p),(q,c))$, si $\forall p,q \in Q$ y $c \in \Gamma$
   $\delta'(((c,q),(q,c)),(a,b)) = q_1$, si $\delta(p,a) = (q,b,\leftarrow)$
   $\delta'(((c,p),(q,c)),(\diamond,b)) = q_2$, si $\delta(p,\llcorner) = (q,b,\leftarrow)$
   $\delta'(q_1,(a,a)) = q_1,\ \forall a \in \Gamma$

5. $F = \{q_1, q_2\}$.

Por lo tanto, la relación de transición de M es automática ya que es reconocida por M'. □

Por otra parte, un *autómata linealmente acotado* es una máquina de Turing con una cantidad limitada de memoria, específicamente por kn celdas, donde n es la longitud de la cadena de entrada w y k es una constante (factor lineal) asociada al autómata. A diferencia de las máquinas de Turing, estos autómatas sólo pueden resolver problemas que requieren memoria que pueda caber dentro de la cinta utilizada para la cadena entrada w.

2.7. Teoría de la computabilidad

En la teoría de la computabilidad se analizan los *problemas indecidibles o irresolubles*, es decir, aquellos que no pueden ser resueltos por un *algoritmo* (programa que siempre termina). Algunos de estos problemas fueron descubiertos durante primera mitad del siglo XX por matemáticos como Kurt Gödel, Alan Turing y Alonzo Church. Algunos problemas irresolubles que se han analizado son:

1. El problema del dominó (The Domino Problem): En este problema se considera que un dominó es un conjunto finito de láminas cuadradas del mismo tamaño (fichas de un dominó), etiquetadas con un número (o color) elegido de un conjunto finito de números (o colores), donde de cada ficha se tiene un número ilimitado de copias. Además, se considera que una teselación del plano es una manera de colocar las fichas del dominó para cubrir todo el plano de tal forma que dos dichas adyacentes compartan el mismo número (o color) en el lado común. Entonces, lo que se busca es decidir si existe una teselación del plano infinito, rayado a cuadros del tamaño del domino, que use esas fichas. Es posible ensamblar las fichas de dominó en un plano infinito, rayado a cuadros del tamaño de un dominó, bajo ciertas reglas establecidas.

2. El problema de la palabra para grupos (The Word Problem for Groups): Sea G un grupo finitamente presentado. Lo que se busca es determinar si existe un algoritmo que decida si dos palabras en los generadores representan el mismo elemento. Equivalentemente, lo que se busca es determinar si existe un algoritmo que determine si una palabra w en los generadores de G es o no la identidad en G. [8, 45]

3. El problema del paro (The Halting Problem): Dado un programa P y una entrada E, se debe decidir si P parará con la entrada E.

Los problemas indecidibles condujeron al desarrollo de modelos teóricos de computadoras, los cuales eventualmente ayudaron a la construcción de las computadoras digitales actuales. Uno de estos modelos fue la máquina de Turing, con la cual Alan Turing demostró que el problema del paro es indecidible.

A continuación se dá una demostración alternativa, sin el uso de máquinas de Turing, de que el problema del paro es irresoluble. La prueba se realiza por contradicción, por lo cual se va a suponer que este problema tiene solución, es decir, que existe un algoritmo que lo resuelve.

Entonces, sea `Paro(P,E)` el algoritmo que resuelve el problema del paro, el cual recibe como entradas al programa `P` y a la entrada `E`, y que responde `Si` cuando el programa `P` se detiene ante la entrada `E` y responde `No` en caso contrario. Haciendo uso de él se construye un algoritmo `X(P)` que recibe como entrada al programa `P` y contiene las siguientes instrucciones:

```
X(P){
 String E;
 E:=Copy(P);
 if( Paro(P,E) ){
    while(true);
 }else{
    return true;
 }
}
```

Este algoritmo genera una copia de `P` en `E` y los pasa como parámetros de entrada al algoritmo `Paro(P,E)`, que es llamado como subrutina. Si `Paro(P,E)` indica que `P` para ante `E`, `X(P)` entra a un ciclo infinito mediante la instrucción `while`. En caso de que `Paro(P,E)` no termine, `X(P)` regresa verdadero. En la figura 2.17 se muestra el diagrama de bloques de lo que realiza la función `X(P)`.

Entonces, el algoritmo X obtiene dos resultados posibles:

$$\texttt{X(P)} = \begin{cases} \text{No para cuando P para ante sí mismo} \\ \text{Para y responde sí cuando P no para ante sí mismo} \end{cases}$$

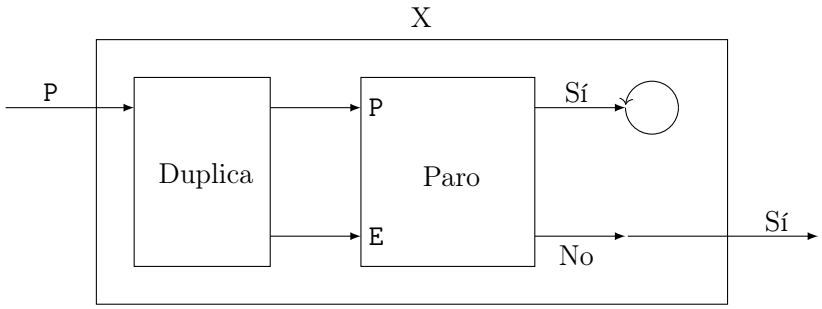


Figura 2.17: Diagrama de bloques del algoritmo `X` que resuelve el problema del paro en la demostración por contradicción.

Siendo así, `X(P)` *no para* cuando la función `Paro(P,E)` responde que `P` para ante sí mismo (ya que `E` es una copia de `P`); mientras que `X(P)` *para y responde sí* cuando `Paro(P,E)` responde que `P` no para ante sí mismo.

Ahora supóngase que `X()` recibe como entrada al mismo programa `X`, es decir `X(X)`. Entonces, por la definición anterior sólo se tienen opciones:

1. `X(X)` *no para* cuando `X` para ante sí mismo, y
2. `X(X)` *para y responde que sí* cuando `X` no para ante sí mismo.

Nótese que ambas opciones se contradicen, lo cual es resultado de haber supuesto que el algoritmo `Paro(P,E)` existía. Por lo tanto, `Paro(P,E)` no existe y el problema del paro no tiene solución, es decir, es irresoluble.

Capítulo 3

Configuraciones con estampa de tiempo

En este capítulo se extiende el concepto de las configuraciones de una máquina de Turing M. Ahora se considera a una cadena $t \in \mathtt{N}$ para representar la estampa de tiempo, la cual está concatenada al inicio de alguna configuración s, formando cadenas del tipo ts. Recuérdese que $\mathtt{N}$ es el conjunto de todas las cadenas binarias que representan a los números naturales $\mathbb{N}$.

Entonces, se probará que dada cualquier $t' = t + 1$ y cualquier transición $s \vdash s'$ de M, la transición $ts \vdash t's'$ es aceptada por un autómata M', es decir, que M' acepta a la convolución de tales pares, $\otimes(ts, t's')$. Nótese que podría suceder que $|t| = |t|$ o que $|t| < |t'|$.

3.1. Definición

Una *configuración con estampa de tiempo* es una cadena $d = ts = tuqv \in \mathtt{N}\Gamma^*Q\Gamma^*$, donde t es una cadena binaria que representa el tiempo transcurrido de un cálculo dado y $s = uqv$ es una configuración (o descripción instantánea) de una máquina de Turing M. Entonces la configuración con estampa de tiempo inicial para M, con una cadena de entrada w, es $0q_0w$, donde el cero denota la representación del tiempo cero en binario y q_0 al estado inicial de M. Se denota $ts \vdash t's'$ (o $d \vdash d'$) cuando $s \vdash s'$ y $t' = t + 1$; en tal caso se dice que d' es *sucesor* de d y que d es *predecesor* de d'.

Tomando en cuenta la notación para las descripciones instantáneas de una máquina de Turing, se denota por $d \overset{n}{\vdash} d_1$, $n \geq 0$, si d transita a d_1 en n pasos $(d \vdash \cdots \vdash d_1)$, es decir, en n unidades de tiempo. Si no es relevante mencionar el número de pasos con los que se transitó de d a d_1, siempre y cuando $n \geq 0$, se denota $d \overset{*}{\vdash} d_1$. En este caso d_1 es *descendiente* de d y d es *ascendiente* de d_1.

De aquí en adelante se supondrá, sin pérdida de generalidad, que Q, Γ y $\{0, 1\}$ serán conjuntos mutuamente disjuntos. Esta convención permitirá identificar inequívocamente las partes que componen a las configuraciones con estampa de tiempo

$d = tuqv$.

3.2. Relación de transición en configuraciones con estampa de tiempo

La relación de transición entre configuraciones con estampa de tiempo también es automática, ya que el autómata M_3 que la reconoce se puede formar usando los autómatas 2.9 y 2.16 (denotados de aquí en adelante por M_1 y M_2, respectivamente). Recuérdese que M_1 reconoce la relación del sucesor, mientras que M_2 reconoce a la relación de transición para configuraciones, sin estampa de tiempo, (ver teorema 2.6.4).

Específicamente M_3 está compuesto por una copia del autómata M_1 y por un número finito de copias del autómata M_2. Esto permite aceptar a las convoluciones $\otimes(d, d')$, si $d \vdash d'$, con la distinción de dos posibles casos: cuando $|t| = |t'|$ y cuando $|t| < |t'|$, tal que $t' = t+1$. Para mostrar que sucede en ambos casos, considérese sin pérdida de generalidad a las transiciones $110u_1q_iu_2v_1v_2 \vdash 111u_1u_2q_jv_1v_2$ y $111u_1q_iu_2v_1v_2 \vdash 1000u_1u_2q_jv_1v_2$. Al igual que en la sección anterior, se escribirán las configuraciones involucradas en cada transición, una debajo de otra (ver tablas 3.1 y 3.2). Nótese que como en la sección anterior, cada i-ésimo par ordenado de las convoluciones $\otimes(110u_1q_iu_2v_1v_2, 111u_1u_2q_jv_1v_2)$ y $\otimes(111u_1q_iu_2v_1v_2, 1000u_1u_2q_jv_1v_2)$ corresponde a los símbolos que están en la i-ésima posición de su tabla respectiva, con la excepción de que en la tabla 3.2 en la posición 9, no existe el símbolo $\diamond$.

Posición	1	2	3	4	5	6	7	8
	1	1	0	u_1	q_i	u_2	v_1	v_2
	1	1	1	u_1	u_2	q_j	v_1	v_2

Tabla 3.1: Caso en donde $|t| = |t+1|$ en una transición de configuraciones con estampa de tiempo. La misma longitud en t permite que las columnas (correspondientes a cada posición) tengan símbolos pertenecientes a cadenas del mismo lenguaje, ya sea en $\mathtt{N}$ o en $\Gamma^*Q\Gamma^*$.

La tabla 3.1 muestra el caso cuando $|t| = |t+1|$. En ella se observa que los lenguajes $\mathtt{N}$ y $\Gamma^*Q\Gamma^*$ se mantienen en columnas separadas, ya que en las tres primeras posiciones, hay una cadena de $\mathtt{N}$, mientras que los caracteres de las posiciones restantes forman una cadena del lenguaje $\Gamma^*Q\Gamma^*$. Entonces, cada par ordenado de $\otimes(110u_1q_iu_2v_1v_2, 111u_1u_2q_jv_1v_2)$ está formado sólo por símbolos de $\{0,1\}$ o de $\Gamma \cup Q$, permitiendo que los pares de la convolución correspondientes a la estampa de tiempo t sean reconocidos por M_1 y el resto de ellos, correspondientes a la parte de la configuración s, por el autómata M_2. Para que esto sea posible, $M_1 = (Q_1, (\Sigma_{1\diamond})^2, q_1^0, \delta_1, F_1)$

Posición	1	2	3	4	5	6	7	8	9
	1	1	1	u_1	q_i	u_2	v_1	v_2	
	1	0	0	0	u_1	u_2	q_j	v_1	v_2

Tabla 3.2: Caso en donde $|t| < |t+1|$ en una transición de configuraciones con estampa de tiempo. La distinta longitud en t provoca un ligero desfasamiento de símbolos en las columnas (correspondientes a cada posición), de tal manera que en la posición 4 $u_1 \in \Gamma$ y $0 \in \{0, 1\}$.

y de $M_2 = (Q_2, (\Sigma_{2\diamond})^2, q_2^0, \delta_2, F_2)$ deben de funcionar de manera conjunta como un único autómata (ver figura 3.1) con de las siguientes modificaciones:

1. q_1^0 es el estado inicial de M_3.
2. F_2 es el conjunto de estados finales de M_3.
3. q_2^0 sigue siendo un estado en M_3 al igual que el conjunto de estados F_1.
4. Se añaden nuevas transiciones que permitan llegar de los estados q_1^1 y q_1^5 de M_1 a algunos estados de M_2 (ver definición de δ_3, inciso b, del lema 3.2.1). Ahora el estado q_1^4 es un estado muerto porque el par $(\diamond, 0)$ no se presenta en las convoluciones $\otimes(ts, t's')$ cuando $ts \vdash t's'$.

Por otra parte, la tabla 3.2 muestra el caso $|t| < |t+1|$, cuando $ts = 111u_1u_2q_iv_1v_2 \vdash 1000u_1u_2v_1q_jv_2 = t's'$. En el segundo renglón se observa que $t+1$ sólo tiene un dígito más que t, lo cual provoca que los símbolos $s'_i \in s'$ estén desfasados sólo una posición a la derecha y que por lo tanto también estén desplazados en cada par ordenado k, donde $5 \leq k \leq 9$. Nótese que M_2 no sabe procesar a los pares de las k-ésimas posiciones, pero que si sabe procesar a los pares (s_{k-1}, s'_k) que se forman mediante $\otimes(s, s')$. También se tiene que observar que el par $(u_1, 0)$ marca el inicio del desfasamiento y que ni M_1 ni M_2 saben procesarlo ya que u_1 y 0 pertenecen a alfabetos disjuntos correspondientes a los lenguajes $\Gamma^*Q\Gamma^*$ y $\mathtt{N}$ respectivamente. Por estas razones M_3 no puede aceptar a la convolución $\otimes(ts, t's')$, si $ts \vdash t's'$, de la misma manera en que lo hace cuando $|t| = |t+1|$. Para estos casos con desfasamiento, M_3 tendrá las siguientes adecuaciones (ver figura 3.2):

1. Se añaden $n = |(\Gamma \cup Q)_\diamond|$ copias del autómata M_2 que son denotadas por $M_2[s_i]$, tal que $s_i \in (\Gamma \cup Q)_\diamond$. Los estados de cada copia están re-etiquetados por (q_2^i, s_i), $q_2^i \in Q_2$, para permitir recordar el símbolo s_i en el siguiente par ordenado y así simular que M_3 tenga lectura de los pares (s_{k-1}, s'_k) de $\otimes(ts, t's')$ después del par que marca el inicio del desfasamiento. En cada copia $M_2[s_i]$ el estado (q_2^0, s_i) no es inicial, mientras que sus estados $F_2 \times (\Gamma \cup Q)_\diamond$ sí son finales.

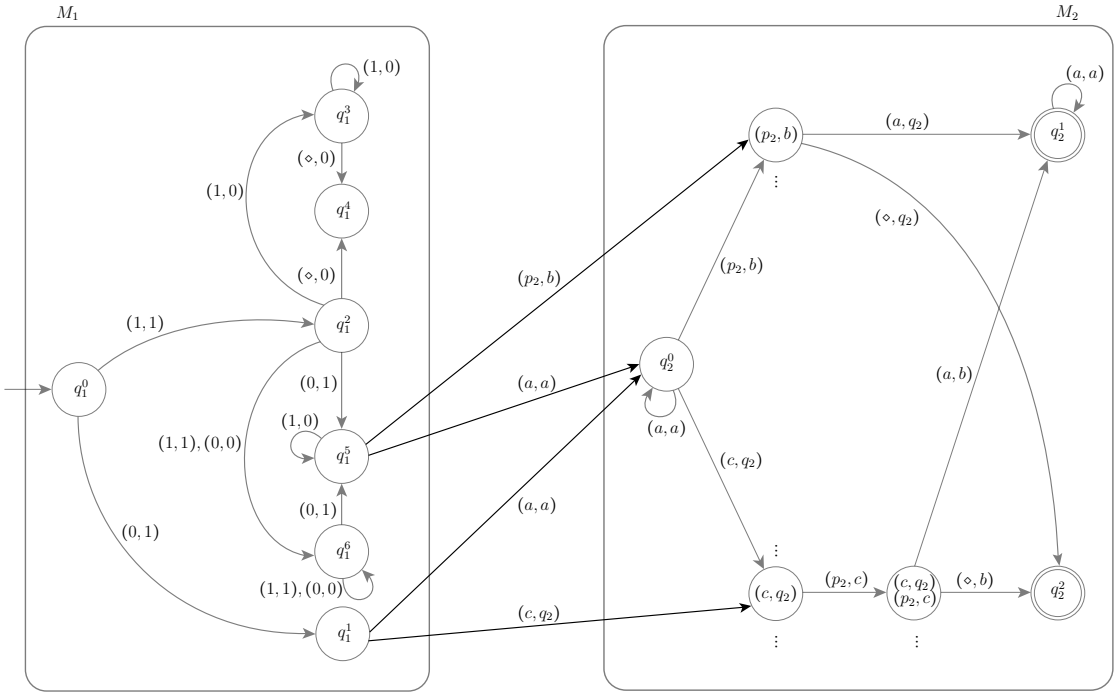


Figura 3.1: Parte del autómata M_3 que reconoce a las configuraciones con estampa de tiempo, cuando $|t| = |t+1|$. Esta está formada por una copia de los autómatas 2.9 y 2.16 (denotados por $M_1 = (Q_1, (\Sigma_{1\diamond})^2, q_1^0, \delta_1, F_1)$ y $M_2 = (Q_2, (\Sigma_{2\diamond})^2, q_2^0, \delta_2, F_2)$, respectivamente). Sólo el estado q_1^0 es estado inicial en M_3; de igual manera, sólo los estados del conjunto F_2 son finales.

2. Se añaden transiciones de los estados $q_1^2, q_1^3 \in M_1$ a ciertos estados de todas las copias $M_2[s_i]$ que permiten leer a los pares ordenados que inician el desfasamiento, (ver definición de δ_3, inciso c, del lema 3.2.1). Por ejemplo, en la tabla 3.2 al leer el par $(u_1, 0)$ se transita a el estado $(q_2^p, u_1) \in M_2[u_1]$.

3. Se añaden transiciones entre la copias $M_2[s_i]$ para simular las transiciones que ocurren originalmente en M_2 y así procesar a los pares de la convolución $\otimes(s, s')$ después del par que marca el desfasamiento en $\otimes(ts, t's')$, (ver definición de δ_3, inciso e, del lema 3.2.1). Entonces, si el autómata M_3 se encuentra en el estado (q_2^k, s_i) y lee el par (s_j, s'_j), éste transita al estado $(\delta_2(q_2^k, (s_i, s'_j)), s_j)$. Nótese que $\delta_2(q_2^k, (s_i, s'_j))$ está permitiendo el procesamiento de los pares $\otimes(s, s')$ por medio del símbolo s_i. Por ejemplo, en la tabla 3.2 cuando M_3 lee el par (u_2, u_1) se encuentra en el estado $(q_2^k, u_1) \in M_2[u_1]$ por lo que transitará al estado $(q_2^r, u_2) \in M_2[u_2]$ si $\delta_2(q_2^k, (u_1, u_1)) = q_2^r$.

Este diseño de M_3 conlleva el siguiente lema:

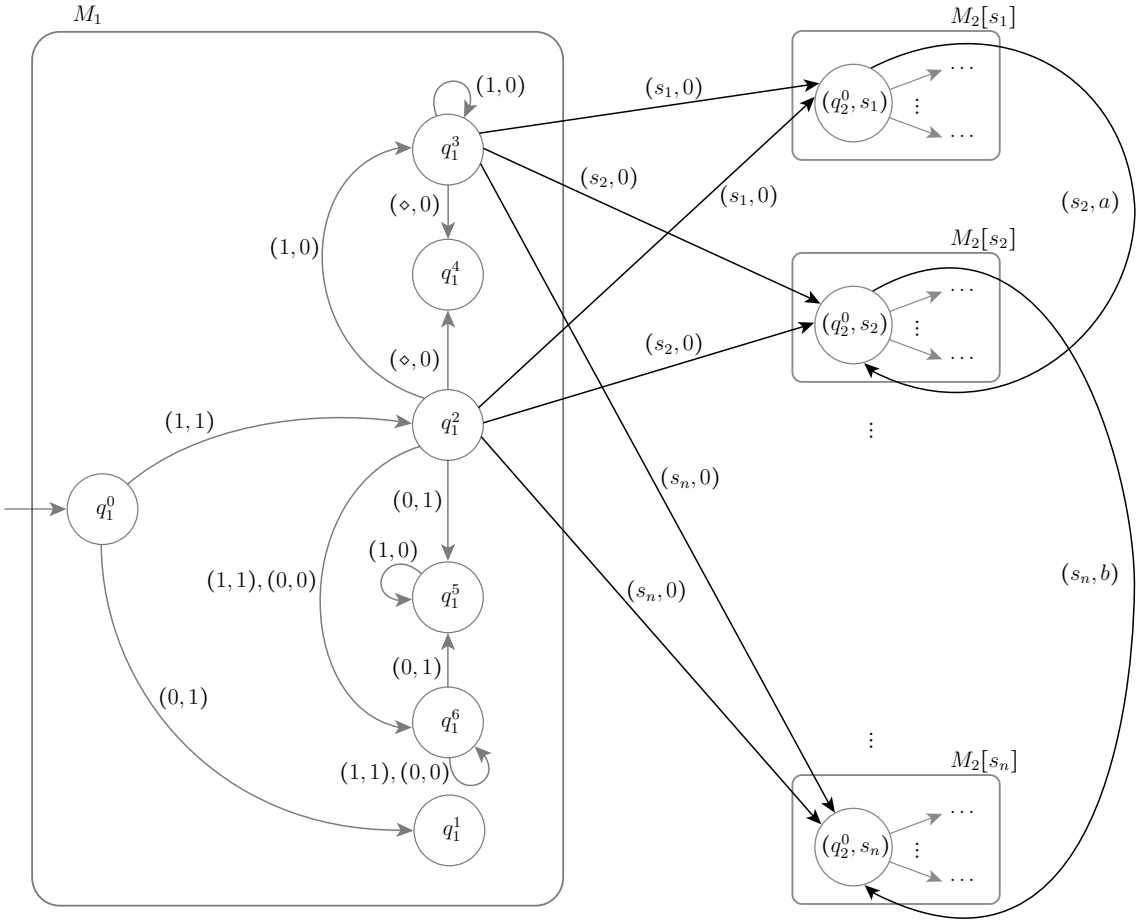


Figura 3.2: Parte del autómata M_3 que reconoce a las configuraciones con estampa de tiempo, cuando $|t| < |t+1|$. Esta está formada por la misma copia de M_1 de la parte anterior (ver imagen 3.1) y por $n = |(\Gamma \cup Q)_\diamond|$ copias del autómata M_2 (denotadas por $M_2[s_i]$, $s_i \in (\Gamma \cup Q)_\diamond$. Solamente el estado q_1^0 es estado inicial en M_3; de igual manera sólo los estados del conjunto $\{F_2 \times (\Gamma \cup Q)_\diamond\}$ son finales. Las flechas curvas en color negro permiten transitar entre las n copias de M_2 para simular la lectura de $\otimes(s, s')$, si $ts \vdash t's'$.

Lema 3.2.1. *La relación de transición de configuraciones con estampa de tiempo* $tuqv \vdash t'u'q'v'$ *es automática para toda máquina de Turing.*

Demostración. Sea $M_1 = (Q_1, (\Sigma_{1\diamond})^2, q_1^0, \delta_1, F_1)$ el autómata que reconoce la convolución de las cadenas binarias de la relación del sucesor $\{(t, t+1) \mid t \in \mathtt{N}\}$ (ver figura 2.9) y $M_2 = (Q_2, (\Sigma_{2\diamond})^2, q_2^0, \delta_2, F_2)$ el autómata que acepta a las transiciones $s \vdash s'$ de una máquina de Turing, es decir, acepta las convoluciones $\otimes(s, s')$ cuando $ts \vdash t's'$, ver figura 2.16. Recuérdese que por hipótesis, $\Sigma_1 \cap \Sigma_2 = \varnothing$. Entonces el autómata finito determinista M_3 que reconoce a las transiciones $ts \vdash t's'$, tal que $t' = t + 1$, se define como $M_3 = (Q_3, \Sigma_3, q_1^0, \delta_3, F_3)$, tal que:

1. $Q_3 = Q_1 \cup Q_2 \cup (Q_2 \times (\Gamma \cup Q)_\diamond)$,
2. $\Sigma_3 = (\Sigma_1 \cup \Sigma_2)^2_\diamond$,
3. q_1^0 es el estado inicial,
4. δ_3:
   - *a)* $\delta_3(q, (a, b)) = \delta_1(q, (a, b))$, si $q \in Q_1$, a y $b \in \Sigma_1$
   - *b)* $\delta_3(q, (a, b)) = \delta_2(q_2^0, (a, b))$, si $q \in F_1$, a y $b \in \Sigma_2$
   - *c)* $\delta_3(q, (a, b)) = (q_2^0, a)$, si $q \in Q_1$, $a \in \Sigma_2$, $b \in \Sigma_1$ y $\delta_1(q, (\diamond, b)) \in F_1$
   - *d)* $\delta_3(q, (a, b)) = \delta_2(q, (a, b))$, si $q \in Q_2$, a y $b \in (\Gamma \cup Q)_\diamond$
   - *e)* $\delta_3((q, a), (b, c)) = (\delta_2(q, (a, c)), b)$, si $q \in Q_2$, a, b y $c \in (\Gamma \cup Q)_\diamond\}$
   - *f)* El resto de las transiciones van a un estado muerto.
5. $F_3 = F_2 \cup (F_2 \times (\Gamma \cup Q)_\diamond)$.

□

Las transiciones que δ_3 realiza corresponden a los siguientes movimientos en M_3:

Inciso	Movimiento
a)	Movimientos en M_1
b)	Paso a M_2 cuando $\lvert t\rvert = \lvert t+1\rvert$
c)	Paso a $M_2[s_i]$ cuando $\lvert t\rvert < \lvert t+1\rvert$
d)	Movimientos en M_2
e)	Movimientos entre las copias $M_2[s_i]$

Tabla 3.3: Descripción de los movimientos en $\delta_3 \in M_3$ definidos en el lema 3.2.1

Obsérvese que ya que las transiciones en una máquina de Turing $M = (Q, \Sigma, \Gamma, \delta, q_0, \llcorner, F)$ son especificadas por la función (parcial) de transición $\delta : Q \times \Gamma \to Q \times \Gamma \times \{\leftarrow, \rightarrow\}$,

el número de predecesores de una configuración d siempre es finita: $|\{d_0 : d_0 \vdash d\}| \leq |Q||\Gamma|$. Entonces, gracias a las estampas de tiempo, el número de antecesores de una configuración d también es finita, ya que si t es la etiqueta de tiempo de d, se tiene que $|\{d_0 : d_0 \overset{*}{\vdash} d\}| \leq 1 + |Q||\Gamma| + (|Q||\Gamma|)^2 + \cdots + (|Q||\Gamma|)^t$; por otra parte, el número de sucesores es 0 o 1. En particular para cualesquiera dos descendientes d_1 y d_2 de d se tiene que $d_1 \overset{*}{\vdash} d_2$ o $d_2 \overset{*}{\vdash} d_1$.

Por cuestiones de simplificación, en lo que sigue se nombraran a las configuraciones con estampa de tiempo simplemente como *configuraciones* de una máquina de Turing.

Capítulo 4

Indecibilidad del K-comportamiento para gráficas automáticas

La indecibilidad de la clan-convergencia ha sido conjeturada implícitamente por varios expertos, comenzando en [48] y en [52], y la primera conjetura explícita de este problema, así como la primera publicación formal, aparecieron respectivamente en [44] y [40]. El principal objetivo de esta línea de investigación es probar que el problema de decidir la clan-convergencia es indecidible para la clase de gráficas finitas (o dar un algoritmo que decida la clan-convergencia). Este problema aún sigue estando abierto.

Sin embargo, en este capítulo se muestra un avance en esta área de investigación, probando que el problema de la clan-convergencia para la clase de gráficas automáticas (que incluye gráficas infinitas) es indecidible. La idea principal de la demostración es reducir el problema del paro al problema de la clan-convergencia simulando las configuraciones de una máquina de Turing con gráficas iteradas de clanes. Entonces, a cualquier MT que pare le corresponderá una secuencia de gráficas de clanes, $G, K^2(G), K^4(G), \ldots$, que convergerá y a cualquier máquina de Turing que no pare, le corresponderá una secuencia que diverja.

Los resultados de este capítulo fueron publicados en [11].

4.1. La indecibilidad de la clan-convergencia

Las gráficas clan-Helly sin vértices dominados son clan-convergentes incluso en el caso infinito (ver teorema 2.2.2), por lo tanto, el siguiente teorema de indecibilidad para gráficas *cuasi-clan-Helly* (aquellas gráficas que se convierten en clan-Helly después de eliminar sólo un vértice) con a lo más un vértice dominado es algo inesperado.

Teorema 4.1.1. *La clan-convergencia es algorítmicamente indecidible para la clase de gráficas automáticas. Además el problema permanece indecidible si la clase se*

reduce a aquellas gráficas que sean cuasi-clan-Helly con grado acotado y que a lo más tengan un vértice dominado.

La demostración se obtiene por la reducción del problema del paro (el cual es indecidible, ver sección 2.7) al problema de la clan-convergencia, de hecho el anterior teorema se sigue inmediatamente del siguiente:

Teorema 4.1.2. *Existe una función computable $\lambda : \mathcal{TMW} \to \mathcal{AG}$ tal que para cada $(M, w) \in \mathcal{TMW}$, M para ante la entrada w si y sólo si $\lambda(M, w)$ es clan-convergente. Por lo tanto, la clan-convergencia es indecidible para cualquier clase que contenga $\mathcal{AG}_0 := \lambda(\mathcal{TMW})$. Además, λ puede ser elegida de tal manera que las gráficas en $\mathcal{AG}_0$ sean cuasi-clan-Helly, de grado acotado y con a lo más un vértice dominado.*

En esta sección se dedicará a demostrar el teorema 4.1.2.

Para reducir el problema del paro al problema de clan-convergencia, se requiere plantear una función computable $\gamma : \mathcal{TMD} \to \mathcal{AG}$ tal que para toda descripción instantánea $d \vdash d'$, tengamos $K^2(\gamma(M, d)) \cong \gamma(M, d')$. Es decir, γ tiene que hacer conmutar el siguiente diagrama:

$$\begin{array}{ccc} \mathcal{TMD} & \xrightarrow{\vdash} & \mathcal{TMD} \\ \downarrow{\scriptstyle \gamma} & & \downarrow{\scriptstyle \gamma} \\ \mathcal{AG} & \xrightarrow{K^2} & \mathcal{AG} \end{array}$$

Primero se diseñará una función auxiliar que sea computable, $\gamma_0 : \mathcal{TM} \to \mathcal{AG}$, y luego $\gamma(M, d)$ se obtendrá agregando a lo más un vértice (y sus adyacencias) a $\gamma_0(M)$. Para definir a γ_0 se considerará una máquina de Turing $M = (Q, \Sigma, \Gamma, \sigma, q_0, \sqcup, F)$ y el conjunto de todas las configuraciones con estampa de tiempo de M, $D = \{tuqv : t \in \mathtt{N}; u, v \in \Gamma^*; q \in Q\}$. Recuérdese que D se puede considerar como una digráfica (ver observación 2.6.2).

Definición 4.1.3. *Ahora se definirá $\gamma_0 : \mathcal{TM} \to \mathcal{AG}$. Dada $M \in \mathcal{TM}$, el conjunto de vértices de $\gamma_0(M)$ es la siguiente concatenación de conjuntos regulares:*

$$V(\gamma_0(M)) = \{1, 2, 3, 4, 5, 6\} \cdot \mathtt{N} \cdot \Gamma^* \cdot Q \cdot \Gamma^*$$

donde un vértice $xd \in V(\gamma_0(M))$ es la concatenación de una etiqueta $x \in \{1, 2, 3, 4, 5, 6\}$ y una configuración con estampa de tiempo $d \in D$. Dos vértices xd y x_1d_1 son adyacentes en $\gamma_0(M)$ si y sólo si se tiene cualquiera de las siguientes condiciones:

1. $d = d_1$ *y* $\{x, x_1\} \in \{\{1, 2\}, \{1, 4\}, \{2, 3\}, \{2, 5\}, \{3, 6\}, \{4, 5\}, \{5, 6\}\}$
2. $d \vdash d_1$ *y* $(x, x_1) \in \{(1, 1), (2, 1), (2, 2), (3, 1), (3, 2), (3, 3)\}$
3. $d_1 \vdash d$ *y* $(x_1, x) \in \{(1, 1), (2, 1), (2, 2), (3, 1), (3, 2), (3, 3)\}$

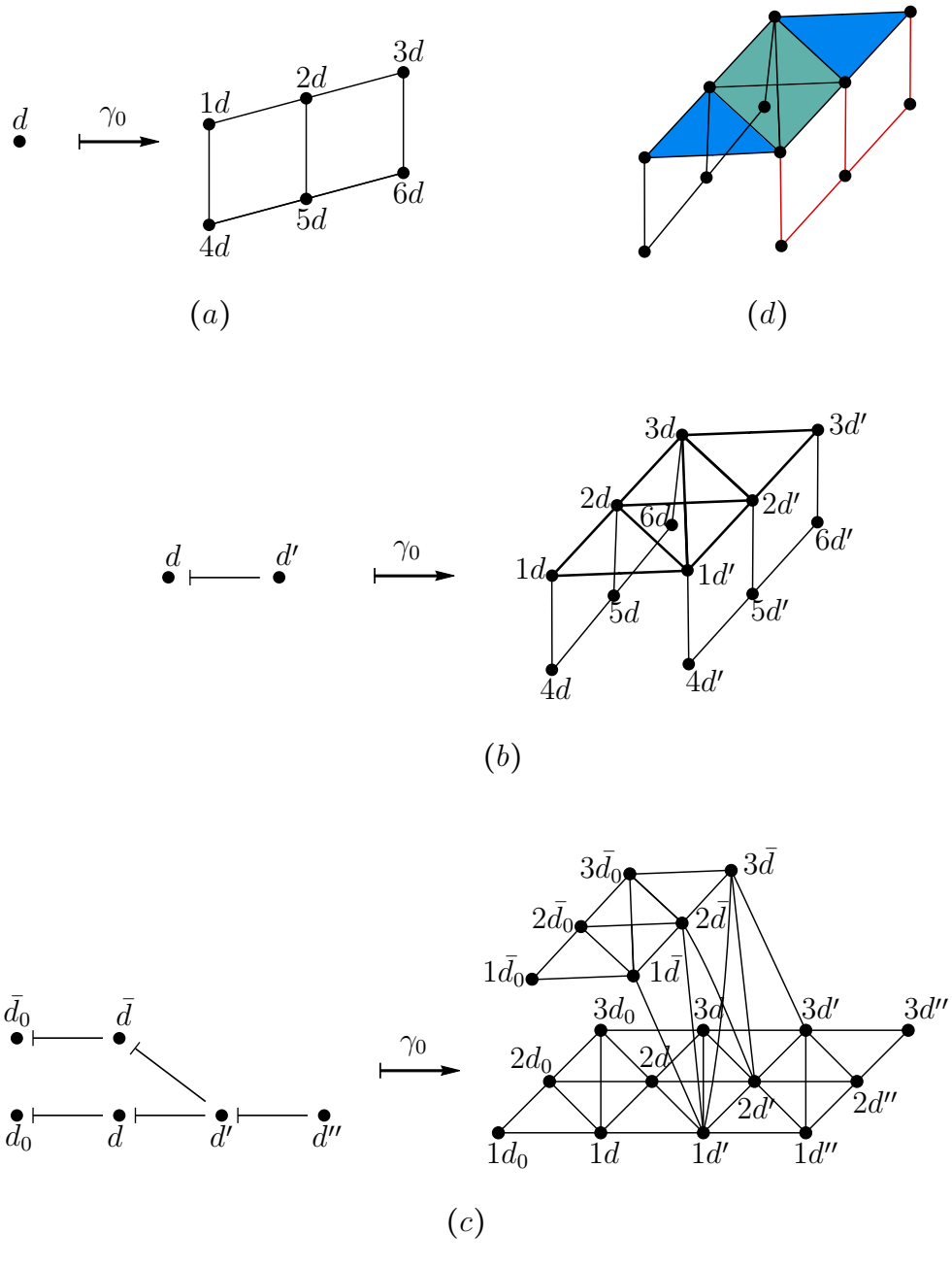


Figura 4.1: Construcción de $\gamma_0(M)$. Los vértices de la forma $4d$, $5d$ y $6d$ se omiten en (c).

La función $\gamma_0(M)$ recién definida genera por cada $d \in D$, seis vértices y siete aristas (ver figura 4.1 (a)) cuyos vértices están etiquetados con $1d, 2d, \ldots, 6d$. Estas etiquetas son cadenas sobre el alfabeto que se genera por la unión del alfabeto de las configuraciones con estampa de tiempo y del conjunto $\{1, 2, \ldots, 6\}$, es decir, sobre $\{0, 1, 2, \ldots, 6\} \cup Q \cup \Gamma$. Además, $\gamma_0(M)$ agrega seis aristas adicionales por cada flecha

$d \vdash d'$ de D (ver figura 4.1 (b)). Por ejemplo, si la digráfica que se muestra a la izquierda de la figura 4.1 (c) fuese un fragmento (subdigráfica) de D, entonces se produciría el fragmento de $\gamma_0(M)$ que se ve en a la derecha de la figura 4.1 (c), sólo que aquí, para tener una mayor claridad del dibujo, los vértices de la forma $4d$, $5d$ y $6d$ son omitidos.

De aquí en adelante, se omitirán los vértices de la forma $4d$, $5d$ y $6d$ de los dibujos, pero eso no significa que los vértices puedan ser eliminados de la gráfica $\gamma_0(M)$ ya que están presentes para evitar la existencia de vértices dominados (y así usar el teorema 2.2.2) y para evitar isomorfismos no deseados (ver teorema 4.1.14).

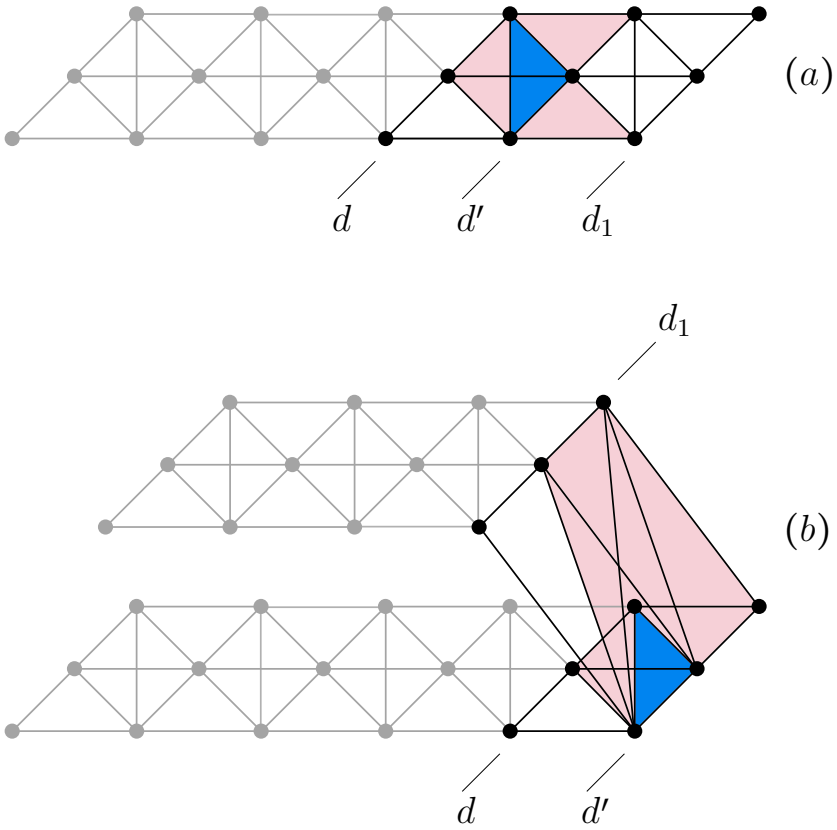


Figura 4.2: Dos fragmentos de la gráfica $\gamma_0(M)$ cuando d es no terminal (de las dos posibles maneras). Los vértices y aristas en color gris pueden estar o no estar presentes en la cinta. En (a) se presenta el caso $d' \vdash d_1$ y en (b) el caso $d_1 \vdash d'$. Para ambos casos, el triángulo extendido del triángulo azul está sombreado de color rosa y tiene como ápice al vértice $2d'$. Los vértices de la forma $4d$, $5d$ y $6d$ son omitidos en los dibujos.

La función γ_0 es computable ya que no se necesita tomar una $d \in D$ a la vez para producir los vértices y aristas iterativamente (lo cual tomaría un tiempo infinito),

sino que sólo se tiene que dar dos autómatas finitos, uno que reconozca a los vértices y otro que reconozca las aristas. Para lo cual, recuérdese que los vértices son fácilmente descritos por una expresión regular y que la relación de transición entre configuraciones (con estampa de tiempo) de una máquina de Turing es automática. Por lo tanto, se obtiene la siguiente observación:

Observación 4.1.4. *La función $\gamma_0 : \mathcal{TM} \to AG$ es una función computable.*

Definición 4.1.5. *Una configuración (con estampa de tiempo) d es no terminal si existe d' y $d_1 \neq d$ tal que $d \vdash d'$, y $d' \vdash d_1$ o $d_1 \vdash d'$ (ver figura 4.2). En otro caso es terminal (ver figura 4.3).*

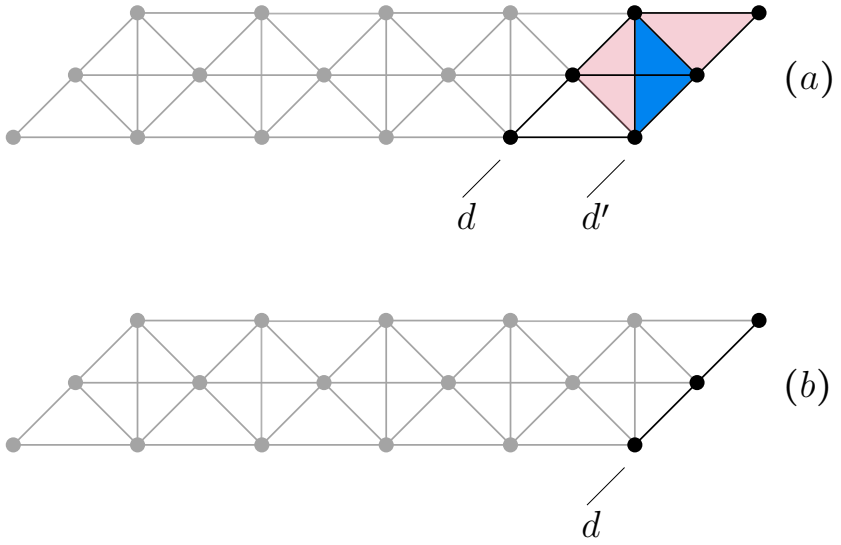


Figura 4.3: Dos fragmentos de la gráfica $\gamma_0(M)$ cuando d es terminal (de las dos posibles maneras). Los vértices y aristas en color gris pueden estar o no estar presentes en la cinta. En el caso (a), el triángulo extendido del triángulo azul está sombreado de color rosa y tiene como ápices a los vértices $3d'$ y $2d'$. Los vértices de la forma $4d$, $5d$ y $6d$ son omitidos en los dibujos.

Nótese que si d es terminal, entonces d o bien d' (cuando $d \vdash d'$) es una configuración de paro y que además cualquier configuración de paro d es terminal.

Ahora se definirá a la función $\gamma : \mathcal{TMD} \to AG$ (ver figura 4.4 (a)): $\gamma(M, d)$ que es casi lo mismo que $\gamma_0(M)$. De hecho, se define $\gamma(M, d) := \gamma_0(M)$ cuando d es terminal. De lo contrario $\gamma(M, d)$ se obtiene de $\gamma_0(M)$ agregando exactamente un vértice (específicamente $0d$, ver figura 4.4 (a)) y sus adyacencias:

Definición 4.1.6. *Sea $(M, d) \in \mathcal{TMD}$. Si d es terminal, se define $\gamma(M, d) := \gamma_0(M)$. De lo contrario, ($d$ es no terminal), se define $V(\gamma(M, d)) := V(\gamma_0(M)) \cup \{0d\}$; en*

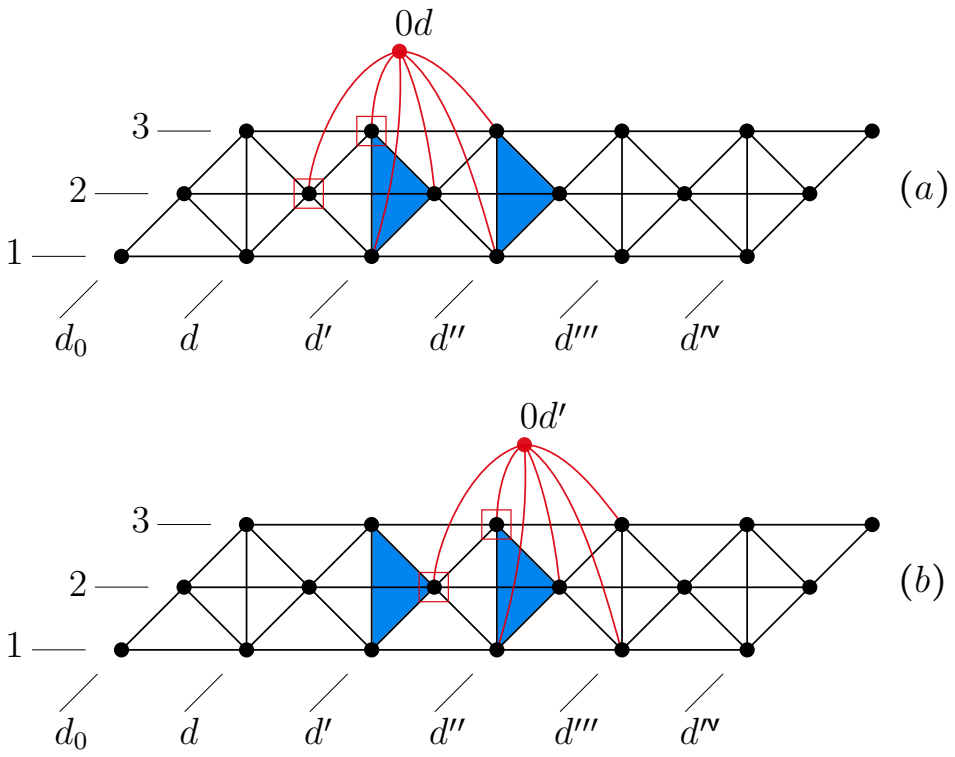


Figura 4.4: (a) y (b) muestra a $\gamma(M, d)$ y $\gamma(M, d')$, respectivamente. Los vértices de la forma $4d$, $5d$ y $6d$ son omitidos en los dibujos.

este caso existe alguna d' con $d \vdash d'$ y las adyacencias en $\gamma(M,d)$ son las mismas que en $\gamma_0(M)$, excepto que los vecinos de $0d$ están dados por: $N(0d) := \{x\bar{d} : x \in \{2,3\}, \bar{d} \vdash d'\} \cup \{1d', 2d', 3d', 1d''\}$ (si no existe d'', tal que $d' \vdash d''$, simplemente se elimina $1d''$ de $N(0d)$).

Los cuadros rojos en la figura 4.4 están allí para recordar que varios vértices como esos pueden estar presentes en $N(0d)$ cuando $|\{\bar{d} : \bar{d} \vdash d'\}| > 1$. Obsérvese que $N(0d)$ es exactamente el triángulo extendido de $\{3d, 1d', 2d'\}$ en $\gamma_0(M)$ (el triángulo azul que está a la izquierda en la figura 4.4 (a)).

Por lo tanto, trayectorias en D como $d \vdash d' \vdash d'' \vdash \cdots \vdash d^{\blacktriangledown}$ producen cintas en $\gamma_0(M)$ como la de figura 4.5. Por Escalante se sabe que ese tipo de cintas son K^2-invariantes (excepto que las cintas de Escalante eran siempre circulares y no tenían vértices de la forma $4d, 5d$ y $6d$) y que ciertas perturbaciones locales en ellas, como el vértice rojo en la figura 4.4 (a), son *perturbaciones viajeras*, en el sentido que la segunda gráfica de clanes de aquella que se muestra en la figura 4.4 (a) es isomorfa a la gráfica de la figura 4.4 (b). Escalante propuso sólo cintas circulares, mientras que las cintas que se generan por $\gamma_0(M)$ son más complejas ya que permiten tener muchas ramificaciones; además también pueden tener "cabos sueltos", por ejemplo, cuando una trayectoria en D termina en $d^{\blacktriangledown}$ por ejemplo, es decir, cuando $d^{\blacktriangledown}$ es una configuración de paro de una máquina de Turing M y por lo tanto $d^{\blacktriangledown}$ no tiene un sucesor en D. En el último caso, las perturbaciones viajeras simplemente desaparecen al final de la cinta correspondiente (ver lema 4.1.12).

Observación 4.1.7. *Si d es terminal, entonces $\gamma(M,d) = \gamma_0(M)$. Si d no es terminal, entonces $\gamma(M,d)$ es obtenido de $\gamma_0(M)$ más la adición de exactamente un vértice (llamado $0d$) y sus adyacencias, como en la figura 4.4 (a).*

Las modificaciones que se le hicieron a $\gamma_0(M)$ para obtener $\gamma(M,d)$ son finitas (un vértice y un número finito de aristas), por lo cual $\gamma(M,d)$ es automática ya que las relaciones finitas son automáticas y la disyunción de relaciones también lo son (ver teorema 2.5.1).

Observación 4.1.8. *$\gamma : \mathcal{TMD} \to \mathcal{AG}$ es una función computable.*

Nótese que cuando está presente $0d$ siempre es dominado por $2d'$ (tal que $d \vdash d'$) y que no hay otro vértice en $\gamma(M,d)$ que sea dominado como se muestra en lema 4.1.9. Para esto se usará el morfismo estrella. Ahora se asume que la concatenación toma precedencia sobre la estrella, de modo que $xd^* = (xd)^*$ y $xd'^* = (xd')^*$.

Lema 4.1.9. *La gráfica $\gamma_0(M)$ es clan-Helly sin vértices dominados. En particular, el morfismo estrella $* : \gamma_0(M) \to K^2(\gamma_0(M))$ es un isomorfismo y para toda $n \geq 0$, $K^{2n}(\gamma_0(M)) \cong \gamma_0(M)$.*

Demostración. Por construcción (ver figura 4.1), todo triángulo en $\gamma_0(M)$ está contenido en un conjunto de la forma $\{xd_1 : x \in \{1,2,3\}$ y $d_1 \in \{d, d'\}\}$ para algunas d y

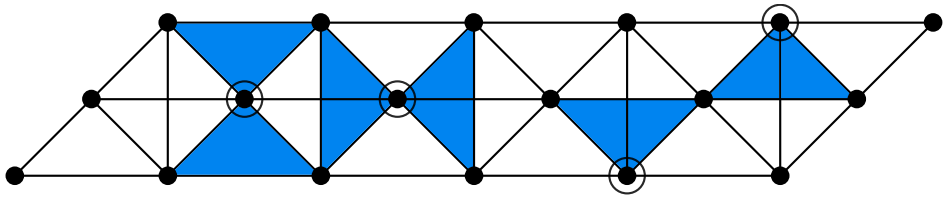


Figura 4.5: Tipos de triángulos de $\gamma_0(M)$. Los vértices de la forma $4d$, $5d$ y $6d$ son omitidos del dibujo.

d', tal que $d \vdash d'$. De esto se sigue que todos los triángulos de $\gamma_0(M)$ son de alguna de las siguientes formas:

$$\{1d, 2d, 1d'\}, \{2d, 3d, 1d'\}, \{2d, 3d, 2d'\}, \{2d, 1d', 2d'\}, \{3d, 1d', 2d'\}, \{3d, 2d', 3d'\}$$

Por inspección directa, se puede ver que los ápices de los correspondientes triángulos extendidos son $2d$, $2d$, $1d'$, $3d$, $2d'$ y $2d'$, respectivamente: en la figura 4.5, para cada triángulo, el ápice de su correspondiente triángulo extendido ha sido marcado con un círculo (algunos triángulos comparten el ápice); nótese que esos vértices son ápices de sus triángulos extendidos independientemente de la multiplicidad de los predecesores de d y d', e independientemente de la existencia de un sucesor de d'. Entonces, por el teorema 2.2.1, $\gamma_0(M)$ es clan-Helly.

No hay vértices dominados en $\gamma_0(M)$ ya que si existiera $xd_1 \neq yd_2$, tal que $N[xd_1] \subseteq N[yd_2]$, entonces $N(xd_1)$ sería un cono y por lo tanto su vecindad sería conexa. Sin embargo (por los vértices de la forma $4d$, $5d$ y $6d$) todo vértice de $\gamma_0(M)$ tiene una vecindad disconexa.

Se sigue que $K^2(\gamma_0(M)) \cong \gamma_0(M)$ por el teorema 2.2.2. □

Recuérdese que el número de predecesores de cualquier configuración es a lo más $|Q||\Gamma|$ y que el número de sucesores es a lo más uno. Ya que $\gamma(M, d)$ se obtiene de $\gamma_0(M)$ añadiendo (a lo más) un vértice y un número finito de aristas, se tiene lo siguiente:

Observación 4.1.10. *$\gamma(M, d)$ es cuasi-clan-Helly con a lo más un vértice dominado y de grado acotado.*

Lema 4.1.11. *Si $d \overset{*}{\vdash} d_1$ y $d \overset{*}{\vdash} d_2$, tal que $d_1 \neq d_2$ y d_1 es no terminal, entonces $\gamma(M, d_1) \ncong \gamma(M, d_2)$.*

Demostración. Supongamos, para llegar a una contradicción, que existe un isomorfismo $\alpha : \gamma(M, d_1) \to \gamma(M, d_2)$. Dado que d_1 es no terminal, $0d_1$ es un vértice de $\gamma(M, d_1)$ y es único vértice dominado de $\gamma(M, d_1)$. Ya que los vértices dominados van a vértices dominados bajo cualquier isomorfismo, se sigue que $0d_2$ también debe de ser un vértice de $\gamma(M, d_2)$ y que $\alpha(0d_1) = 0d_2$.

Ahora, si se toma a d_1' y d_2', tal que $d_1 \vdash d_1'$ y $d_2 \vdash d_2'$, entonces el vértice $2d_1' \in \gamma(M, d_1)$ es el único ápice de $N(0d_1)$ y $2d_2' \in \gamma(M, d_2)$ es el único ápice de $N(0d_2)$; de ésto se sigue que $\alpha(2d_1') = 2d_2'$.

Para cualquier $\bar{d}_1$ tal que $\bar{d}_1 \vdash d_1'$, se tiene que $\{2\bar{d}_1 : \bar{d}_1 \vdash d_1'\}$ es exactamente el conjunto de vértices de $N(0d_1)$ que tienen grado tres (en $N(0d_1)$) y que pertenecen a una completa de cuatro vértices (de manera análoga sucede en $\gamma(M, d_2)$). Entonces, se sigue que el conjunto $\{2\bar{d}_1 : \bar{d}_1 \vdash d_1'\}$ es mapeado biyectivamente por α sobre $\{2\bar{d}_2 : \bar{d}_2 \vdash d_2'\}$. Similarmente, cuando $\bar{d}_1$ es un antecesor de d_1' se tiene que $\alpha(2\bar{d}_1) = 2\bar{d}_2$ para algún antecesor $\bar{d}_2$ de d_2', es decir, que $\{2\bar{d}_1 : \bar{d}_1 \overset{*}{\vdash} d_1'\}$ es mapeado biyectivamente por α sobre $\{2\bar{d}_2 : \bar{d}_2 \overset{*}{\vdash} d_2'\}$.

Ya que $d \overset{*}{\vdash} d_1 \vdash d_1'$, $d \overset{*}{\vdash} d_2 \vdash d_2'$ y $d_1 \neq d_2$, se sigue que $d_1' \neq d_2'$ y que $d_2' \overset{*}{\vdash} d_1'$ o $d_1' \overset{*}{\vdash} d_2'$; se asumirá la última sin pérdida de generalidad. Se sigue que el número (finito) de antecesores de d_1' es menor que el número (también finito) de antecesores de d_2'. Pero esto contradice la última afirmación del párrafo anterior. □

En el siguiente lema se comenzarán a usar intensamente los clanes de $\gamma(M, d)$ y $\gamma_0(M)$. Por lo cual, es conveniente observar todos los tipos de clanes de $\gamma_0(M)$ que se muestran en la figura 4.1 (d), los cuales están en color azul, verde y rojo. También, cuando $\gamma_0(M)$ está dibujada con en la figura 4.4(a) (suponiendo que $0d$ no está presente), todo clan de al menos tres vértices comienza en su vértice que esté más a la izquierda. Esto permite indexar los clanes (con al menos tres vértices) de $\gamma_0(M)$ (ver figura 4.4 (a)):

$$\begin{aligned} q_{1d} &:= \{1d, 2d, 1d'\}, \\ q_{2d} &:= \{2d, 3d, 1d', 2d'\}, \\ q_{3d} &:= \{3d, 2d', 3d'\}. \end{aligned}$$

Lema 4.1.12. *$d \vdash d'$ implica que $K^2(\gamma(M, d)) \cong \gamma(M, d')$. De lo contrario ($d \nvdash d'$ para cualquier d'), $K^2(\gamma(M, d)) \cong \gamma_0(M) = \gamma(M, d)$.*

Demostración. Durante esta demostración, se recomienda mantener a la figura 4.4 a la vista. Los cuadros rojos en esta figura están allí para recordar que puede haber otros vértices relevantes además de los que aparecen en la figura; por ejemplo, en el caso de la figura 4.4 (a), los vértices extra de la forma $2\bar{d}$ y $3\bar{d}$ existen por cada $\bar{d}$ que satisface $\bar{d} \vdash d'$, con $\bar{d} \neq d$. Esos vértices extras son adyacentes a $0d$.

Primero se supondrá que d y d' no son terminales. Por el lema 4.1.9, $\gamma_0(M)$ es clan-Helly sin vértices dominados y existe un isomorfismo $* : \gamma_0(M) \to K^2(\gamma_0(M))$. Ya que los clanes son estructuras locales y que $\gamma(M, d)$ se obtiene de $\gamma_0(M)$ añadiendo un sólo vértice nuevo, $0d$, se puede analizar las propiedades de $\gamma(M, d)$ en términos de las propiedades de $\gamma_0(M)$ y las modificaciones que el vértice nuevo produce. Obsérvese que en particular, el morfismo estrella $* : \gamma(M, d) \to K^2(\gamma(M, d))$ debe comportarse como un isomorfismo casi en cualquier parte, excepto, alrededor del nuevo vértice $0d$.

Alrededor de $0d$, sólo se producen cambios menores por el nuevo vértice. Unos pocos clanes de $\gamma_0(M)$ son extendidos con el vértice $0d$, a saber: por cada $\bar{d}$ con $\bar{d} \vdash d'$ (incluyendo $\bar{d} = d$) todos los clanes de la forma: $q_{2\bar{d}} := \{2\bar{d}, 3\bar{d}, 1d', 2d'\}$, $q_{3\bar{d}} := \{3\bar{d}, 2d', 3d'\}$ y $q_{1d'} := \{1d', 2d', 1d''\}$; entonces, los clanes extendidos son:

$$\begin{array}{rcccl} \hat{q}_{2\bar{d}} & := & q_{2\bar{d}} \cup \{0d\} & = & \{2\bar{d}, 3\bar{d}, 1d', 0d, 2d'\} \\ \hat{q}_{3\bar{d}} & := & q_{3\bar{d}} \cup \{0d\} & = & \{3\bar{d}, 0d, 2d', 3d'\} \\ \hat{q}_{1d'} & := & q_{1d'} \cup \{0d\} & = & \{1d', 0d, 2d', 1d''\}. \end{array}$$

Estos clanes extendidos no cambian las adyacencias entre ellos (como vértices de $K(\gamma(M,d)))$ ya que todos ellos ya eran adyacentes por el vértice en común $2d'$, ni tampoco los clanes extendidos cambian las adyacencias con otros clanes ya que los demás no contienen al vértice $0d$. Solamente un clan extra aparece, el clan

$$q_{0d} := \{0d, 2d', 3d', 1d''\}.$$

Se sigue que $K(\gamma(M,d))$ puede ser obtenida de $K(\gamma_0(M))$ añadiendo el vértice q_{0d} y sus adyacencias.

Ahora los clanes de clanes de $\gamma(M,d)$ (es decir, los clanes de $K(\gamma(M,d))$, los vértices de $K^2(\gamma(M,d)))$ son en su mayoría los de $\gamma_0(M)$ (los cuales son estrellas de vértices por el teorema 2.2.2), excepto alrededor del vértice $0d$. Obsérvese que alrededor del vértice $0d$, $0d^*$ no es clan de clanes ya que éste no es maximal: $0d^* \subsetneq 2d'^*$, y que las estrellas de los vecinos de $0d$ son modificados remplazando los clanes extendidos antes mencionados cuando sea pertinente; por ejemplo, después de añadir el vértice $0d$, la nueva estrella del vértice $1d'$ se convierte en:

$$1d'^* = \{q_{1\bar{d}} : \bar{d} \vdash d'\} \cup \{\hat{q}_{2\bar{d}} : \bar{d} \vdash d'\} \cup \{\hat{q}_{1d'}, \{1d', 4d'\}\}.$$

Además de estas modificaciones, las estrellas de tres vértices (específicamente: $2d'$, $3d'$ y $1d'')$ son extendidos por la adición del nuevo clan $q_{0d} = \{0d, 2d', 3d'1d''\}$, por ejemplo, después de añadir el vértice $0d$, la estrella de $3d'$ se convierte en:

$$3d'^* = \{\hat{q}_{3\bar{d}} : \bar{d} \vdash d'\} \cup \{q_{0d}, q_{2d'}, q_{3d'}, \{3d', 6d'\}\}.$$

Se concluye que todas las estrellas xd_1^*, para $xd_1 \neq 0d$, están presentes en $K^2(\gamma(M,d))$ y que tienen la misma adyacencia entre ellas como las tenían en $K^2(\gamma_0(M))$; recuérdese que para cualquier gráfica G, $x^*, y^* \in K^2(G)$ implica que $x^* \simeq y^*$ si y sólo si $x \simeq y$.

Ahora considérese el triángulo extendido $\hat{T}$ de $T := \{3d', 1d'', 2d''\}$ (el triángulo azul de la derecha en la figura 4.4 (a)). Un nuevo clan de clanes aparece y sus elementos son precisamente los clanes contenidos en $\hat{T}$, a saber:

$$Q_{0d} := \{q_{2\bar{d}'} : \bar{d}' \vdash d''\} \cup \{q_{0d}, q_{3d'}, q_{1d''}\}.$$

Nótese que cuando d''' no existe en D, tampoco el vértice $q_{1d''}$ existe en $K(\gamma(M,d))$ (en este caso, simplemente se omite $q_{1d''}$ en Q_{0d}), pero bajo la suposición que se hizo

al inicio de la demostración, d' no es terminal, entonces Q_{0d} siempre es un clan nuevo de $K(\gamma(M,d))$.

Es fácil ver que no hay otro clan además de los ya considerados. Para esto supongamos que $Q = \{q_1, q_2, \ldots, q_r\}$ es otro clan de clanes, entonces se debe tener que $\cap Q = \varnothing$ y $q_{0d} \in Q$ (de lo contrario se tendría $Q = xd^*$ para algún $xd \in \gamma_0(M,d)$). Ya que $2d' \notin \cap Q$, se sigue que Q contiene al menos uno de $q_{3d'}$, $q_{1d''}$ o $q_{2\bar{d}'}$ (para algún $\bar{d}' \neq d'$ con $\bar{d}' \vdash d''$). Ahora las condiciones $3d', 1d'' \notin \cap Q$ implican que Q debe contener a $q_{3d'}$ y ya sea a $q_{1d''}$ o $q_{2\bar{d}'}$ para algún $\bar{d}' \vdash d''$ con $\bar{d}' \neq d'$. Cualquier clan de clanes sujeto a estas condiciones debe satisfacer $Q \subseteq Q_{0d}$ y por lo tanto $Q = Q_{0d}$.

Por consiguiente, $K^2(\gamma(M,d))$ puede ser obtenido de $K^2(\gamma_0(M)) \cong \gamma_0(M)$ agregando el vértice Q_{0d} y sus adyacencias. Ahora, nótese que $\cup Q_{0d} = \{xd' : x \in \{2,3\}, \bar{d}' \vdash d''\} \cup \{0d, 1d'', 2d'', 3d'', 1d'''\} = N_{\gamma(M,d')}(0d') \cup \{0d\}$. Entonces, para $xd_1 \neq 0d$ (recuérdese que $0d^*$ no es clan de clanes) las adyacencias de Q_{0d} en $K^2(\gamma(M,d))$ están dadas por $Q_{0d} \sim xd_1^* \Leftrightarrow \exists q,\ q \in Q_{0d} \cap xd_1^* \Leftrightarrow \exists q,\ xd_1 \in q \in Q_{0d} \Leftrightarrow xd_1 \in \cup Q_{0d} \Leftrightarrow xd_1 \in N_{\gamma(M,d')}(0d') \Leftrightarrow xd_1 \sim 0d'$ en $\gamma(M,d')$. Por lo tanto, se sigue que $K^2(\gamma(M,d)) \cong \gamma(M,d')$.

Ahora, si d' es terminal y d es no terminal, entonces el vértice $0d'$ en $\gamma(M,d')$ por definición no existe en $\gamma(M,d')$ y tampoco el vértice Q_{0d} existe en $K^2(\gamma(M,d))$ ya que en este caso (después de la eliminación de los clanes inexistentes) se tiene que $Q_{0d} \nsubseteq 3d'^*$; por lo tanto, $K^2(\gamma(M,d)) \cong \gamma_0(M) = \gamma(M,d')$. Pero si d' y d son terminales, es decir, cuando $\gamma(M,d) = \gamma_0(M) = \gamma(M,d')$, entonces $K^2(\gamma(M,d)) = K^2(\gamma_0(M)) \cong \gamma_0(M) = \gamma(M,d')$.

Finalmente, si $d \nvdash d'$ para cualquier d', donde d es terminal y $\gamma(M,d) = \gamma_0(M)$, se tiene que $K^2(\gamma(M,d)) = K^2(\gamma_0(M)) \cong \gamma_0(M) = \gamma(M,d)$.

□

Lema 4.1.13. *Para toda d y $a \geq 0$ existe alguna d_1 con $d \overset{*}{\vdash} d_1$ tal que $K^{2a}(\gamma(M,d)) \cong \gamma(M,d_1)$.*

Demostración. Si existe alguna d_1 tal que $d \overset{a}{\vdash} d_1$, entonces por la aplicación iterada del teorema 4.1.12, se tiene que $K^{2a}(\gamma(M,d)) \cong \gamma(M,d_1)$.

De lo contrario, existe una $b < a$ y alguna d_1 con $d \overset{b}{\vdash} d_1$ tal que $d_1 \nvdash d_2$ para cualquier d_2. En este caso d_1 es terminal y $\gamma(M,d) = \gamma_0(M)$. Por lo tanto $K^{2a}(\gamma(M,d)) = K^{2a-2b}(K^{2b}(\gamma(M,d))) = K^{2a-2b}(\gamma(M,d_1)) \cong K^{2a-2b}(\gamma_0(M)) \cong \gamma_0(M) = \gamma(M,d_1)$. □

Lema 4.1.14. *Si dada alguna $n < m$ sucede que $K^n(\gamma(M,d)) \cong K^m(\gamma(M,d))$, entonces para alguna $a < b$ se tiene que $K^{2a}(\gamma(M,d)) \cong K^{2b}(\gamma(M,d))$.*

Demostración. Por el lema 4.1.13 se tiene que para cada número natural c, existe algún d_1 tal que $d \overset{*}{\vdash} d_1$ y $K^{2c}(\gamma(M,d)) \cong \gamma(M,d_1)$. Esta gráfica tiene muchos vértices cuyas vecindades son isomorfas a una gráfica sin aristas en tres vértices (los de la forma $5d_2 \in V(\gamma(M,d_1))$, para cualquier d_2). Un análisis directo de los clanes

de $\gamma(M, d_1)$ (ver figura 4.1 (d)) y sus intersecciones muestra que $K^{2c+1}(\gamma(M,d)) \cong K(\gamma(M,d_1))$ no tiene tales vértices. Entonces, $K^n(\gamma(M,d)) \cong K^m(\gamma(M,d))$ es sólo posible cuando n y m tienen la misma paridad. Por lo tanto, $K^{2a}(\gamma(M,d)) \cong K^{2b}(\gamma(M,d))$ ya sea para $2a = n < m = 2b$ o para $2a = n+1 < m+1 = 2b$. □

Lema 4.1.15. *Si para alguna $a < b$ se tiene que $K^{2a}(\gamma(M,d)) \cong K^{2b}(\gamma(M,d))$, entonces:*

$$K^{2a}(\gamma(M,d)) \cong \gamma_0(M) \cong K^{2b}(\gamma(M,d)).$$

Demostración. Por el lema 4.1.13, existen d_1 y d_2 con $d \overset{*}{\vdash} d_1$, $d \overset{*}{\vdash} d_2$ tal que $K^{2a}(\gamma(M,d)) \cong \gamma(M,d_1)$ y $K^{2b}(\gamma(M,d)) \cong \gamma(M,d_2)$.

Si d_1 o d_2 fueran terminales, se tendría que $\gamma(M,d_1) = \gamma_0(M)$ o que $\gamma(M,d_2) = \gamma_0(M)$ y el resultado se sigue.

Si d_1 y d_2 son no terminales, entonces no pueden ser iguales porque las configuraciones deben ser diferentes: si $d = ts$, $d_1 = t_1 s_1$ y $d_2 = t_2 s_2$ entonces se necesita que $t_1 = t + a < t + b = t_2$. Por el lema 4.1.11 se tiene que $K^{2a}(\gamma(M,d)) \cong \gamma(M,d_1) \ncong \gamma(M,d_2) \cong K^{2b}(\gamma(M,d))$, lo cual resulta una contradicción. Por lo tanto, d_1 y d_2 tienen que ser terminales y el resultado se sigue como en el párrafo anterior. □

Lema 4.1.16. *M se detiene con una entrada w si y sólo si $\lambda(M,w) := \gamma(M, 0q_o w)$ es clan-convergente.*

Demostración. M se detiene con una entrada $w \Leftrightarrow 0q_0 w \overset{*}{\vdash} d$ para alguna configuración terminal $d \Leftrightarrow K^{2a}(\gamma(M,0,q_0 w)) \cong \gamma(M,d) \cong \gamma_0(M)$ para alguna $a \geq 0 \Leftrightarrow K^{2a}(\gamma(M,0q_0 w)) \cong K^{2b}(\gamma(M,0q_0 w))$ para alguna $b > a \geq 0 \Leftrightarrow \lambda(M,w) := \gamma(M,0q_0 w)$ es clan-convergente. □

Por lo tanto, se concluye que la primera afirmación del teorema 4.1.2 es verdadera por el lema 4.1.16 y la observación 4.1.8. La segunda afirmación de este mismo es un corolario inmediato de la primera y el tercera afirmación se sigue de la observación 4.1.10. Esto concluye la demostración del teorema 4.1.2.

4.2. Consecuencias de la indecibilidad de la clan-convergencia

Ya se ha probado que la clan-convergencia es indecidible para la clase de gráficas automáticas. Varias consecuencias pueden extraerse a partir de eso, por ejemplo, cuando las gráficas que se usaron convergen, lo hacen hacia un 2-ciclo de gráficas clan-Helly (como está definido en [52]: $K^n(G) \cong K^{n+2}(G)$, donde ambas, $K^n(G)$ y $K^{n+1}(G)$ son clan-Helly). Así el problema de decidir si una gráfica es *a la larga clan-Helly* (es decir, $K^n(G)$ es clan-Helly para alguna n) y el problema de decidir si una gráfica es *a la larga* 2*-autóclana* (es decir, $K^n(G) \cong K^{n+2}(G)$ para alguna n) son equivalentes al problema de la clan-convergencia en la clase de gráficas $\mathcal{AG}_0$.

En particular, esos problemas son indecidibles para cualquier clase que contenga a $\mathcal{AG}_0$. El problema de la clan-divergencia también es evidentemente indecidible para gráficas automáticas. Lo mismo es verdadero para el problema del reconocimiento de gráficas eventualmente autóclanas (es decir, $K^n(G) \cong K^{n+1}(G)$ para alguna n):

Teorema 4.2.1. *El problema de decidir si una gráfica es eventualmente autóclana es indecidible para gráficas automáticas.*

Demostración. Se usará libremente la notación y resultados de la sección anterior.

La reducción requerida del problema del paro, $\delta : \mathcal{TMW} \to \mathcal{AG}$ está dada por $\delta(M,w) = \lambda(M,w) \boxtimes K(\lambda(M,w))$. Nótese que $\delta(M,w)$ ciertamente es automática: el producto fuerte de gráficas automáticas es automática y la gráficas de clanes de las gráficas en $\mathcal{AG}_0$ son fácilmente descriptibles como gráficas automáticas y los autómatas correspondientes pueden ser calculados efectivamente a partir de $\lambda(M,w)$.

Es bien sabido que el operador de clanes se distribuye sobre el producto fuerte, ésto es, $K(A \boxtimes B) \cong K(A) \boxtimes K(B)$ (la prueba en [47] es válida para gráficas infinitas). También, las gráficas conexas se factorizan de manera única con respecto a el producto fuerte (hasta repeticiones de factorizaciones triviales). Además, cada componente conexo de $\gamma(M,d)$ contiene vértices (específicamente los de la forma $5d_1$) cuyas vecindades cerradas son isomorfas a $K_{1,3}$ (gráfica bipartita completa con 1 vértice en una parte y 3 vértices en la otra). Ya que $K_{1,3}$ no es factorizable de manera no trivial, entonces $\gamma(M,d)$ tampoco lo es. Lo mismo se puede decir para $K(\gamma(M,d))$ ya que las vecindades cerradas de los clanes de la forma $\{5d_1, 6d_1\}$ en $K(\gamma(M,d))$ tampoco son trivialmente factorizables. Por lo tanto, resulta que la (esencialmente) única factorización de $K^n(\delta(M,w))$, para n par, es $K^n(\delta(M,w)) = K^n(\lambda(M,w) \boxtimes K(\lambda(M,w))) \cong K^n(\lambda(M,w)) \boxtimes K^{n+1}(\lambda(M,w)) \cong \gamma(M,d) \boxtimes K(\gamma(M,d))$ para alguna d, con $0q_0w \overset{*}{\vdash} d$.

Si M se detiene en w, d es terminal para alguna d satisfaciendo $0q_0w \overset{*}{\vdash} d$. Por lo tanto $\gamma(M,d) = \gamma_0(M)$ y

$$\begin{aligned} K^{n+1}(\delta(M,w)) &= K(K^n(\delta(M,w))) \cong K(\gamma(M,d) \boxtimes K(\gamma(M,d))) \\ &\cong K(\gamma_0(M) \boxtimes K(\gamma_0(M))) \cong K(\gamma_0(M)) \boxtimes K^2(\gamma_0(M)) \\ &\cong K(\gamma_0(M)) \boxtimes \gamma_0(M) \cong K^n(\delta(M,w)) \end{aligned}$$

y así, $\delta(M,w)$ es eventualmente autóclana.

Recíprocamente se tiene $K^n(\delta(M,w)) \cong K^{n+1}(\delta(M,w))$ cuando $\delta(M,w)$ es eventualmente autóclana. Se asumirá sin pérdida de generalidad que n es par, entonces $\gamma(M,d) \boxtimes K(\gamma(M,d)) \cong K(\gamma(M,d)) \boxtimes \gamma(M,d_1)$ para alguna d y d_1 con $0q_0w \overset{*}{\vdash} d$, y $d \vdash d_1$ o d como una configuración de paro y $d_1 = d$. Pero esto sólo es posible si d es terminal y por lo tanto si M se detiene en w.

Entonces resulta que M se detiene con w si y sólo si $\delta(M,w)$ es eventualmente autóclana. □

Como ya se había señalado, el teorema 4.1.2 se puede fortalecer de varias maneras, redefiniendo λ para reducir la clase $\mathcal{AG}_0$. Por ejemplo, es fácil modificar la digráfica

de configuraciones (con estampa de tiempo) D para que su ingrado sea a lo más 2 (añadiendo vértices extras). Ésto permite declarar que el problema de la clan-convergencia es indecidible incluso para gráficas automáticas con grado máximo a lo más 10. Similarmente, se puede restringir que las gráficas en $\mathcal{AG}_0$ sean conexas (añadiendo aristas extra entre componentes conexos, pero sin formar triángulos), con exactamente un "punto de salida" (conectando todos los vértices de paro en D con una trayectoria dirigida, quizá infinita en una de sus direcciones), etc.

Gracias al teorema 2.5.2, el teorema 4.1.1 implica que:

Teorema 4.2.2. *La propiedad de la clan-convergencia no es expresable en lenguaje de primer orden para gráficas automáticas.*

Además, en vista de la generalización del teorema 2.5.2 el cual aparece en [31, Teorema 3.2], la propiedad de la clan-convergencia tampoco es expresable cuando se extiende lógica de primer orden para incluir distintos cuantificadores generalizados (incluyendo $\exists^w$, $\exists^{(k,m)}$) e incluso cuando se extiende para incluir la versión restringida de los cuantificadores de segundo orden considerados en FSO.

Capítulo 5

Inexpresibilidad del K-comportamiento en lenguaje de primer orden

El operador de clanes, K, es ampliamente considerado como uno de las más complejos y la caracterización de la K-divergencia ha resistido todos los intentos durante 46 años, ya que la noción de las gráficas iteradas de clanes fue introducida en [25]. Como consecuencia, entre los expertos ha crecido la inquietud de saber si la clan-divergencia es indecidible, pero no se ha tenido un progreso substancial en esta dirección.

En el capítulo anterior se probó que la clan-convergencia es indecidible para la clase de gráficas automáticas (no necesariamente finitas), lo cual implica que la clan-convergencia no es expresable en lenguaje de primer orden para la misma clase. En este capítulo se fortalece dicho corolario al demostrar que la propiedad de la clan-divergencia no es expresable en lenguaje de primer orden incluso para la clase de gráficas finitas. La expresabilidad lógica tiene fuertes relaciones con la teoría de la complejidad [28,43] y consecuentemente, se abren nuevas rutas de investigación para este estudio en el problema de decisión del K-comportamiento.

Los resultados de este capítulo fueron publicados en [13].

5.1. Juegos de Ehrenfeucht-Fraïssé de k-rondas

Dadas un par de gráficas $\mathfrak{A}$ y $\mathfrak{B}$ y un número entero $k \in \mathbb{N}$, el *juego de Ehrenfeucht-Fraïssé de k-rondas* [43, 56] en $\mathfrak{A}$ y en $\mathfrak{B}$ es jugado por dos jugadores *Duplicador* y *Saqueador*: en cada ronda, Saqueador primero selecciona a un vértice x en $\mathfrak{A}$ o en $\mathfrak{B}$ ("él decide") y Duplicador responde seleccionando algún vértice y en la otra gráfica. El ganador es determinado como sigue: después de k-rondas, algunos vértices $a_1, a_2, \ldots, a_k$ son seleccionados en $\mathfrak{A}$ y algunos otros vértices $b_1, b_2, \ldots, b_k$ son seleccionados en $\mathfrak{B}$ (el subíndice indica la ronda en la que cada vértice fue seleccionado,

las repeticiones de vértices es permitida, en este punto no importa cual jugador seleccionó cada vértice), entonces Duplicador gana el juego cuando el mapeo dado por $a_i \mapsto b_i$ es un isomorfismo de la subgráfica de $\mathfrak{A}$ inducida por $\{a_1, a_2, \ldots, a_k\}$ a la subgráfica de $\mathfrak{B}$ inducida por los vértices $\{b_1, b_2, \ldots, b_k\}$; de lo contrario, Saqueador gana el juego. Se dice que Duplicador tiene una *estrategia ganadora* si existe una forma en la cual Duplicador puede jugar para garantizar la victoria después de las k-rondas establecidas, no importa como juegue el Saqueador.

Teorema 5.1.1 (Ehrenfeucht-Fraïssé, Corolario 3.10 en [43])**.** *Una propiedad P de σ-estructuras finitas no es expresable en primer orden si para cada $k \in \mathbb{N}$, existen dos σ-estructuras finitas, $\mathfrak{A}_k$ y $\mathfrak{B}_k$, tal que:*

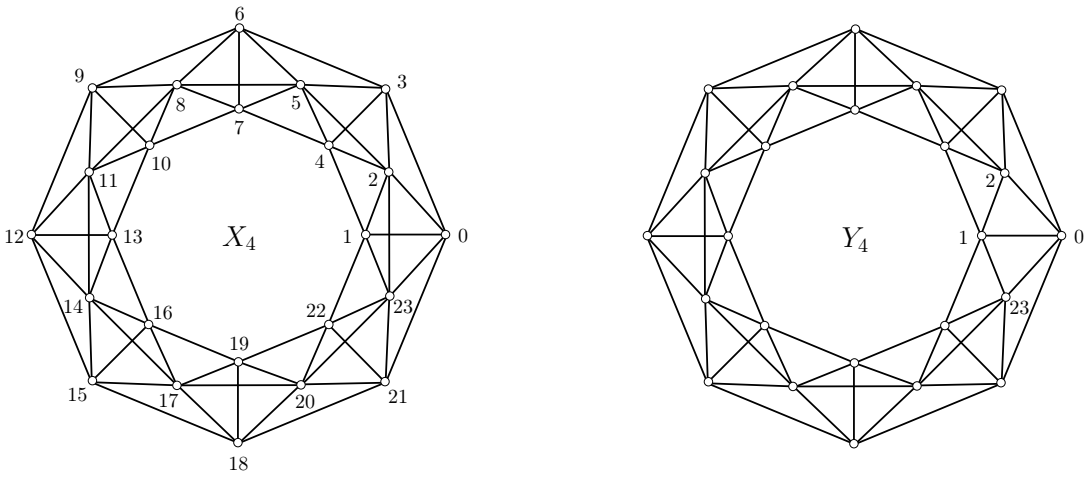
1. *Duplicador tiene una estrategia ganadora de k-rondas para el juego de Ehrenfeucht-Fraïssé en $\mathfrak{A}_k$ y $\mathfrak{B}_k$.*

2. *$\mathfrak{A}_k$ tiene una propiedad P y $\mathfrak{B}_k$ no la tiene.*

5.2. Clanes y juegos de Ehrenfeucht-Fraïssé

De aquí en lo que resta del capítulo se usarán el tipo de letra estándar de LaTeX "mathnormal" $(A, B, \ldots)$ para denotar a las gráficas, a menos que el resultado que se considera sea más un modelo teórico que una gráfica, en cuyo caso se usará el tipo de letra fraktur $(\mathfrak{A}, \mathfrak{B}, \ldots)$.

Considérese las familias de gráficas que se muestran en la figura 5.1:



(a) (b)

Figura 5.1: (a) X_4, (b) Y_4.

Definición 5.2.1. *Sea $m \geq 2$. La gráfica X_m tiene al conjunto de vértices $V(X_m) = \mathbb{Z}_{6m}$ y adyacencias dadas por: $x \sim y$ siempre que $x \equiv 1 \mod 3$ y $y - x \in \{1,2,3\}$, o cuando $x \not\equiv 1 \mod 3$ y $y - x \in \{1,3\}$ (o cuando $y \sim x$, de acuerdo a las reglas anteriores). La gráfica Y_m es obtenida de X_m quitando exactamente una arista: $\{6m-1, 2\} \in E(X_m)$.*

Estas gráficas, X_m y Y_m, pertenecen a la clase de gráficas denominadas relojes [17, 33, 36]. Recuérdese que el clan-comportamiento de los relojes es decidido por un algoritmo que se ejecuta en tiempo polinomial [40]. Para los propósitos de este capítulo, será necesario saber el clan-comportamiento de X_m y Y_m:

Observación 5.2.2. *Para toda $m > 2$, X_m es K-convergente y Y_m es K-divergente.*

Demostración. De acuerdo con la terminología de [40], tanto X_m y Y_m son relojes con vértices no cubiertos, y con cero y un *segmento bueno*, respectivamente. Se sigue por el algoritmo descrito en [40, Teorema 3.6] que $|K^n(X_m)| = |X_m| + 0 \cdot n = 6m$ y que $|K^n(Y_m)| = |Y_m| + 1 \cdot n = 6m + n$. Por lo tanto X_m es K-convergente y Y_m es K-divergente. □

Obsérvese ahora las gráficas en la figura 5.2. Note que las gráficas (a) y (b) no son isomorfas pero hay un par de biyecciones de vértices que casi son isomorfismos: una simple traslación funciona bien excepto en las regiones rojas, mientras una reflexión (seguida de una traslación) funciona bien excepto en las regiones verdes. Esta noción es la que se desea capturar con las definiciones de cuasi-isomorfismos y de gráficas cuasi-isomorfas.

Dadas dos gráficas A y B, un *cuasi-isomorfismo* es una biyección $f : V(A) \to V(B)$ junto con dos *regiones de falla* $X \subseteq V(A)$ y $X' \subseteq V(B)$ tal que siempre que f no sea un isomorfismo en un par de vértices x y y (es decir, cuando $x \sim y$ y $f(x) \not\sim f(y)$ o cuando $x \not\sim y$ y $f(x) \sim f(y)$), se tiene que $x, y \in X$ y $f(x), f(y) \in X'$.

Un par de gráficas $\mathfrak{A}$ y $\mathfrak{B}$ son *cuasi-isomorfas con una distancia de falla s*, si existen dos cuasi-isomorfismos $r : V(A) \to V(B)$ con regiones de falla G y G' y $g : V(A) \to V(B)$ con regiones de falla R y R' tal que $r(G) = G' = g(G)$ y $r(R) = R' = g(R)$ y tal que $d_A(R, G) = s$.

Dado que cualquier camino de longitud mínima de R a G (respectivamente de R' a G') no usa aristas contenidas en G (respectivamente en G'), y ya que r es un isomorfismo salvo por las aristas contenidas en G y G', se debe de tener que $d_B(R', G') = d_A(R, G)$.

Teorema 5.2.3. *Si $\mathfrak{A}$ y $\mathfrak{B}$ son gráficas cuasi-isomorfas con una distancia de falla mayor que 2^k, entonces el Duplicador tiene una estrategia ganadora para el juego de Ehrenfeucht-Fraïssé de k-rondas en $\mathfrak{A}$ y $\mathfrak{B}$.*

Demostración. Asúmase que las biyecciones y las regiones de falla son r, g, G, G', R y R' como anteriormente. Se describirá recursivamente la estrategia del Duplicador:

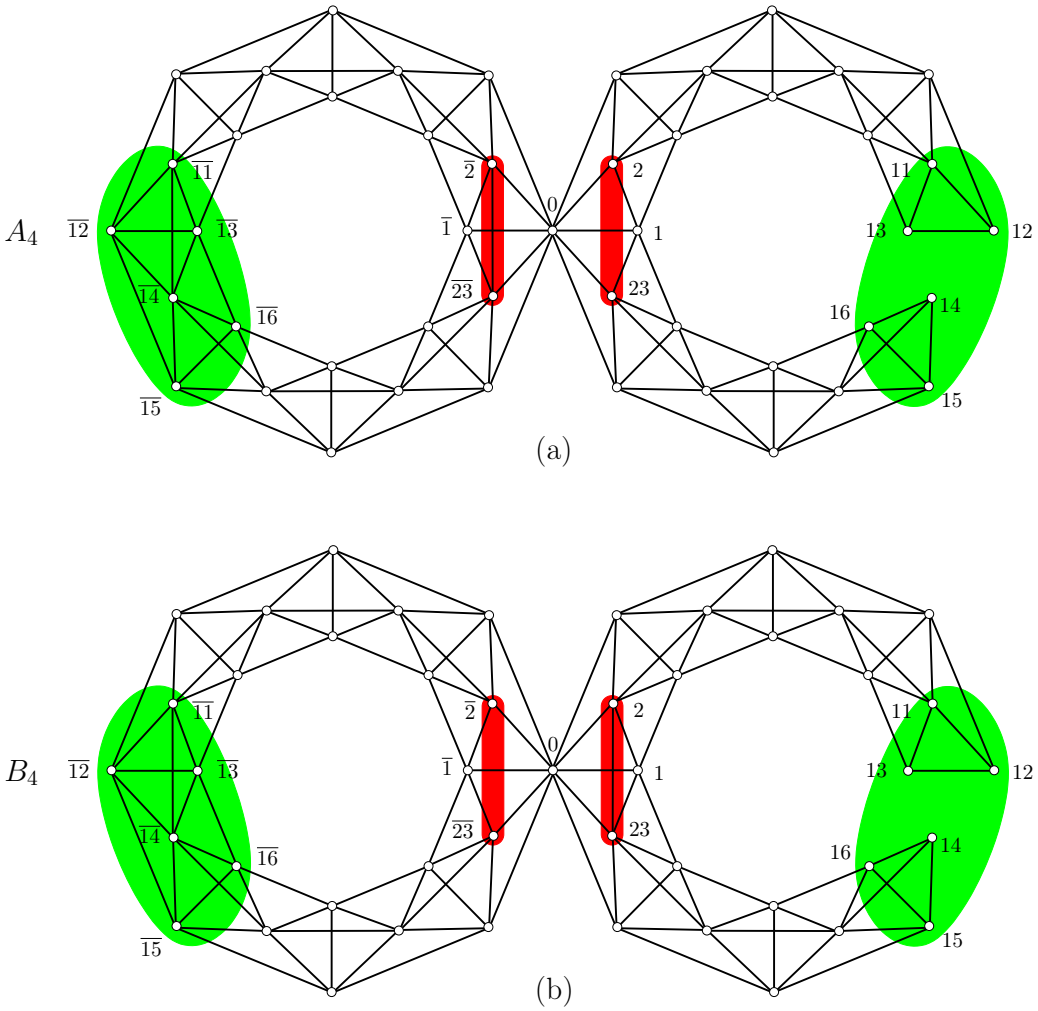


Figura 5.2: (a) $\mathfrak{A}_1 = A_4$, (b) $\mathfrak{B}_1 = B_4$.

En cada ronda i, $1 \leq i \leq k$, habrá regiones rojas y verdes R_i, $G_i \subseteq V(\mathfrak{A})$ y $R'_i, G'_i \subseteq V(\mathfrak{B})$ de los vértices previamente seleccionados (inicialmente se coloca $R_1 = R$, $R'_1 = R', G_1 = G, G'_1 = G'$). Primero asúmase que Saqueador selecciona un vértice $x \in \mathfrak{A}$. La idea es que Duplicador replique con $r(x)$ o $g(x)$ de acuerdo a si x es más cercano a la región roja R_i o a la región verde G_i, es decir, Duplicador selecciona un vértice $dup(x) \in \mathfrak{B}$ de acuerdo a la siguiente regla:

$$dup(x) = \begin{cases} r(x) & \text{si } d_{\mathfrak{A}}(x, R_i) \leq d_{\mathfrak{A}}(x, G_i), \\ g(x) & \text{si } d_{\mathfrak{A}}(x, R_i) > d_{\mathfrak{A}}(x, G_i). \end{cases}$$

Después de eso, se calcula una nueva región roja y verde como sigue:

Si $dup(x) = r(x)$**:** se define $R_{i+1} = R_i \cup \{x\}$, $R'_{i+1} = R'_i \cup \{r(x)\}$, $G_{i+1} = G_i$ y $G'_{i+1} = G'_i$,

Si $dup(x) = g(x)$**:** se define $G_{i+1} = G_i \cup \{x\}$, $G'_{i+1} = G'_i \cup \{g(x)\}$, $R_{i+1} = R_i$ y $R'_{i+1} = R'_i$.

Si en cambio Saqueador seleccionó $x \in \mathfrak{B}$, entonces Duplicador replica análogamente con $r^{-1}(x)$ o $g^{-1}(x)$ de acuerdo a si x es el más cercano a R'_i o G'_i, los nuevos conjuntos R_{i+1}, R'_{i+1}, G_{i+1}, G'_{i+1} también son calculados análogamente en este caso:

Si $dup(x) = r^{-1}(x)$**:** se define $R_{i+1} = R_i \cup \{r^{-1}(x)\}$, $R'_{i+1} = R'_i \cup \{x\}$, $G_{i+1} = G_i$ y $G'_{i+1} = G'$,

Si $dup(x) = g^{-1}(x)$**:** se define $G_{i+1} = G_i \cup \{g^{-1}(x)\}$, $G'_{i+1} = G'_i \cup \{x\}$, $R_{i+1} = R_i$ y $R'_{i+1} = R'_i$.

Se afirma que para $1 \leq i \leq k+1$, $d_{\mathfrak{A}}(R_i, G_i) = d_{\mathfrak{B}}(R'_i, G'_i) > 2^{k-i+1}$: de hecho para $i = 1$ se tiene $d_{\mathfrak{A}}(R_1, G_1) > 2^k$ por hipótesis y $d_{\mathfrak{B}}(R'_1, G'_1) = d_{\mathfrak{A}}(R_1, G_1)$ como se señaló anteriormente. Para el paso de inducción, primero se asumirá que $x \in \mathfrak{A}$ está más cerca a la región roja R_i, entonces

$$2^{k-i+1} < d_{\mathfrak{A}}(R_i, G_i) \leq d_{\mathfrak{A}}(R_i, x) + d_{\mathfrak{A}}(x, G_i) \leq 2d_{\mathfrak{A}}(x, G_i).$$

Se sigue que $d_{\mathfrak{A}}(x, G_i) > 2^{k-i+2} = 2^{k-(i+1)+1}$. Ahora, obsérvese que

$$d_{\mathfrak{A}}(R_{i+1}, G_{i+1}) = min\{d_{\mathfrak{A}}(R_i, G_i), d_{\mathfrak{A}}(x, G_i)\} > 2^{k-(i+1)+1}.$$

También se debe de tener que $d_{\mathfrak{B}}(R'_{i+1}, G'_{i+1}) = d_{\mathfrak{A}}(R_{i+1}, G_{i+1})$ ya que cualquier camino de longitud mínima de la región roja a la verde no usa aristas de adentro de esas regiones y ya que $r : V(\mathfrak{A}) \to V(\mathfrak{B})$ es un isomorfismo fuera de esas regiones, entonces, r mapea los caminos de longitud mínima de R_{i+1} a G_{i+1} en caminos de longitud mínima de R'_{i+1} a G'_{i+1}. Se sigue que $d_{\mathfrak{B}}(R'_{i+1}, G'_{i+1}) = d_{\mathfrak{A}}(R_{i+1}, G_{i+1}) > 2^{k-(i+1)+1}$ en este caso. Los otros tres casos son análogos, a saber: cuando $x \in \mathfrak{A}$ está más cerca de la región verde G_i, y cuando $x \in \mathfrak{B}$ está más cerca de la región roja R_i y de la región verde G_i. Esto concluye la prueba de la afirmación.

La estrategia descrita es una estrategia ganadora para el Duplicador ya que después de la ronda k-ésima, se tiene $d_{\mathfrak{A}}(R_{k+1}, G_{k+1}) = d_{\mathfrak{B}}(R'_{k+1}, G'_{k+1}) > 2^{k-(k+1)+1} = 1$, lo cual significa que las regiones roja y verde nunca están en contacto (es decir, siempre quedan a distancia por lo menos dos) y por lo tanto, hay un isomorfismo de la subgráfica de $\mathfrak{A}$ inducida por los vértices seleccionados a la subgráfica de $\mathfrak{B}$ inducida por los vértices seleccionados: es decir, el isomorfismo que actúa como r en la región roja R_{k+1} y actúa como g en la región verde G_{k+1}. □

Ahora se definirá a las gráficas A_m y B_m (ver figura 5.2 (a) y (b)). Para la construir A_m se toma una copia de X_m y una copia de Y_m, se renombran los vértices de X_m como $\overline{0}, \overline{1}, \ldots, \overline{6m-1}$, ahora se identifica el vértice $\overline{0} \in X_m$ y el vértice $0 \in Y_m$, finalmente, se remueven las aristas que conectan el conjunto $\{3m-1, 3m, 3m+1\}$ al conjunto $\{3m+2, 3m+3, 3m+4\}$; entonces, la gráfica resultante es A_m. Ahora, B_m es obtenida de A_m quitando la arista $\{\overline{2}, \overline{6m-1}\}$ y agregando la arista $\{2, 6m-1\}$.

Teorema 5.2.4. *La clan-divergencia no es expresable en lenguaje de primer orden.*

Demostración. Se define $\mathfrak{A}_k = A_{2^k+2}$ y $\mathfrak{B}_k = B_{2^k+2}$. Obsérvese que la distancia de falla es mayor que 2^k en ambas gráficas: para $k = 1$ $(m = 4)$ se puede ver fácilmente en la figura 5.2, y siempre que m incrementa en 1, la brecha de distancia también incrementa en 1. Se sigue por el teorema 5.2.3 que el Duplicador tiene una estrategia ganadora para el juego de Ehrenfeucht-Fraïssé de k-rondas en $\mathfrak{A}_k$ y $\mathfrak{B}_k$.

También observése que A_m se desmantela a X_m y que B_m se desmantela a Y_m: por ejemplo, en la figura 5.2 (a), los vértices 12 y 13 de A_4 son dominados por el vértice 11, y también el vértice 14 es dominado por el vértice 15. Después de eliminar los vértices $12, 13$ y 14, los vértices $11, 15$ y 16 se vuelven dominados. Claramente el proceso puede continuar hasta que quede una sola copia de X_4 (esto se puede hacer para toda m). Consideraciones similares se aplican a B_m. Se sigue por el teorema 2.2.4 y la observación 5.2.2 que cada A_m es clan-convergente y cada B_m es clan-divergente.

El resultado se sigue del teorema 5.1.1. □

Capítulo 6

Simulación de compuertas lógicas con gráficas de clanes

La dinámica discreta generada por el operador de clanes cuando se aplica de manera iterativa en gráficas, ha mostrado ser muy compleja. Esta complejidad ha fascinado a expertos en el campo de la dinámica de gráficas, así como también ha derrotado todos sus intentos por caracterizar el clan-comportamiento de las gráficas. De hecho, desde el año 2001, Medianis [44] sembró la inquietud sobre el poder computacional del operador de clanes, preguntando si éste es *Turing-completo*, es decir, si es capaz de simular el comportamiento de cualquier máquina de Turing. En el esfuerzo (hasta ahora incompleto) de probar que el operador de clanes es Turing-completo, se han empezado a simular circuitos digitales a través de gráficas de clanes, los cuales son reportados en este capítulo.

Los resultados de este capítulo fueron publicados en [12].

6.1. Idea básica para el diseño de una computadora digital

En electrónica digital es bien sabido que (muchas copias de) las compuertas en un *conjunto funcionalmente completo de compuertas lógicas* (por ejemplo $\{\texttt{AND}, \texttt{NOT}\}$ o $\{\texttt{OR}, \texttt{NOT}\}$ o simplemente $\{\texttt{NAND}\}$ o $\{\texttt{NOR}\}$) es todo lo que se necesita para construir una computadora digital completa [46].

La afirmación del párrafo anterior requiere una serie de supuestos implícitos que pueden no ser tan obvios como podría pensarse, ya que una computadora digital también necesita de otros componentes básicos y más complejos. Por ejemplo, dentro de los componentes básicos están: los cables, portadores de señal, divisores de señal y empalmes.

Para la creación de *flip-flops* (componente esencial para almacenar información en una computadora digital) y otros componentes complejos se necesita contemplar

que el ancho del pulso de la señal eléctrica sea al menos tan largo como el tiempo de conmutación de las compuertas, más el tiempo de propagación de las señales en los cables de los bucles de retroalimentación interna de los componentes.

Con el uso de gráficas y de la dinámica del operador de clanes es posible simular la mayoría de los componentes mencionados anteriormente. Tener una simulación de *todos* los componentes digitales requeridos (aún no logrados), implicaría los siguientes tres hechos:

1. La simulación de una computadora digital completa utilizando el operador de clanes.

2. El operador de clanes tendría al menos el mismo poder computacional que los autómatas linealmente acotados para la clase de gráficas finitas.

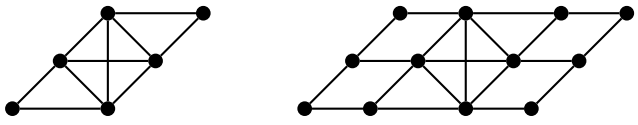
3. El operador de clanes tendría todo el poder computacional de las máquina de Turing para la clase de *gráficas cuasi periódicas* (definidas en la sección 6.3).

En las siguientes secciones se muestran dos codificaciones distintas, pero no ajenas, que permiten la simulación de algunos componentes digitales mediante el uso de gráficas y del operador de clanes.

6.2. Primera codificación

A continuación se muestran los componentes que forman la primera codificación. Se recomienda visitar el sitio web, citado en [10], donde se muestra material suplementario para la visualización de la dinámica del operador de clanes en estos componentes.

El bloque de construcción básico de esta primera codificación se muestra en la figura 6.1 (a). Todos los vértices de este bloque son vértices dominados, los cuales son indeseables para el propósito en cuestión porque tienden a desaparecer cuando se aplica el operador de clanes. En particular, este bloque colapsaría a un sólo vértice después de unas cuantas iteraciones de K. Para evitar esto, se añaden algunos vértices extra como se muestra en la figura 6.1 (b).



(a) (b)

Figura 6.1: (a) Bloque de construcción básico. (b) Bloque de construcción básico sin vértices dominados.

Uniendo varios (tantos como sean necesarios) bloques de construcción se pueden construir *canales* como el que muestra en la figura 6.2, el cual consta de cuatro bloques básicos y de los vértices de los extremos. Estos últimos ayudan a no tener vértices dominados de la misma manera que en la figura 6.1 (b).

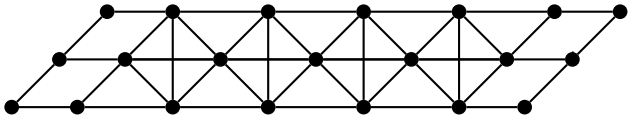


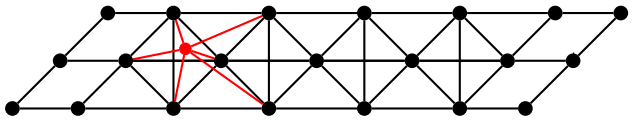
Figura 6.2: Canal.

Dentro de los canales podría haber algunas perturbaciones locales, llamadas *fotones*, las cuales serán representadas en color rojo (ver figura 6.3). Los fotones se mueven a través del canal cuando se aplica el operador de clanes. De hecho, la gráfica de la figura 6.3 (a) es el canal original con su respectiva perturbación (fotón) y las dos gráficas en los inicisos (b) y (c), de la misma figura, son obtenidas al aplicar K^2 y K^4, respectivamente. En lo que resta del capítulo se usarán potencias pares de K debido a que los componentes básicos son (casi) siempre K^2-invariantes pero no K-invariantes, por lo tanto, resulta más fácil ver qué pasa con potencias pares de K porque lo único que cambia son las perturbaciones locales.

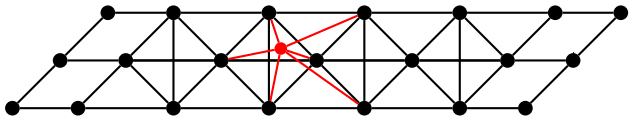
El comportamiento de desplazamiento de los fotones fue observado por primera vez por Escalante en [17] y fue estudiando extensivamente en [33,36,40] por otros más. Con esta teoría ya desarrollada, resulta claro que el comportamiento del operador de clanes mostrado en la figura 6.3 es exactamente el que se acaba de describir en el párrafo anterior. De manera alternativa, este comportamiento se puede verificar por medio de: una computadora, el software GAP + YAGS [9, 22] y el código del apéndice A.

Primera codificación: La presencia y ausencia de un fotón en un canal codifica respectivamente al símbolo 1 y al símbolo 0.

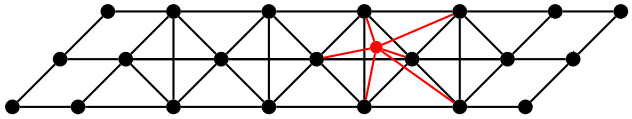
Esta primera codificación hace factible la simulación de la compuerta `OR` mediante gráficas de clanes. Recuérdese que por definición, la compuerta `OR` tiene el comportamiento que se muestra en la tabla 6.1. Cada fila, de arriba a abajo, de esta tabla es simulada respectivamente por la figura 6.4, 6.5, 6.6 y 6.7. En estas tres últimas simulaciones, cada configuración inicial de la compuerta `OR` (en su respectiva entrada) se muestra en el inciso (a), y cada gráfica de los incisos sucesivos se obtiene al aplicar el operador de clanes K^2 a la gráfica del inciso previo.



(a)



(b)



(c)

Figura 6.3: Las subgráficas en color rojo representan a las perturbaciones locales (fotones) de los canales, las cuales se mueven (en este caso, de izquierda a derecha) a causa del operador de clanes. Si G es la gráfica que se muestra en (a), entonces las gráficas de (b) y (c) son $K^2(G)$ y $K^4(G)$, respectivamente.

A	B	Q
0	0	0
0	1	1
1	0	1
1	1	1

Tabla 6.1: Definición de la compuerta `OR`.

Además de la compuerta `OR` otros componentes también pueden ser simulados con las gráficas de clanes, tal como se establece en el siguiente teorema:

Teorema 6.2.1. *Usando las gráficas de clanes y la primera codificación se pueden simular canales, portadores de señal (fotones), compuertas* `OR`*, divisores (ver figura 6.8 y 6.9) y empalmes (ver figuras 6.10, 6.11, 6.12, 6.13).*

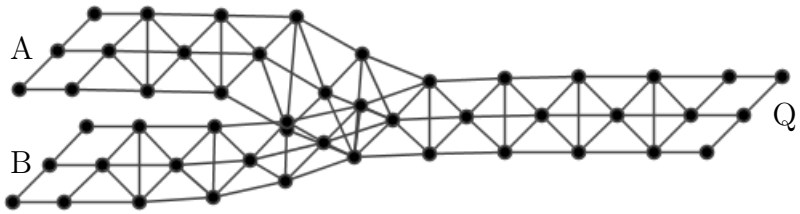


Figura 6.4: Gráfica que representa la configuración de la compuerta `OR` cuando en ambas entradas, A y B, hay un cero (ausencia de fotón). Dado que la gráfica es K^2-invariante en la salida Q no se presentará ninguna perturbación.

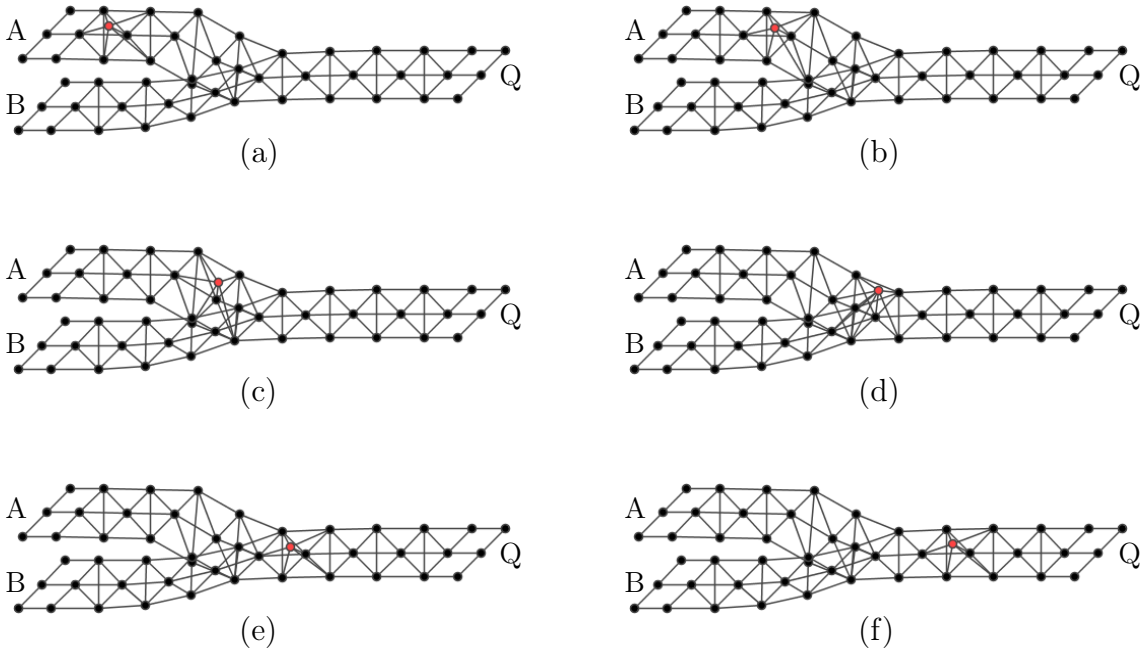


Figura 6.5: La gráfica G en (a) representa la configuración de la compuerta `OR` cuando en A entra un 1 y en B un 0. Las gráficas de (b) hasta (f) son $K^2(G)$ a $K^{10}(G)$, respectivamente. De (b) a (f) se muestra el traslado gradual del fotón hacia la salida Q.

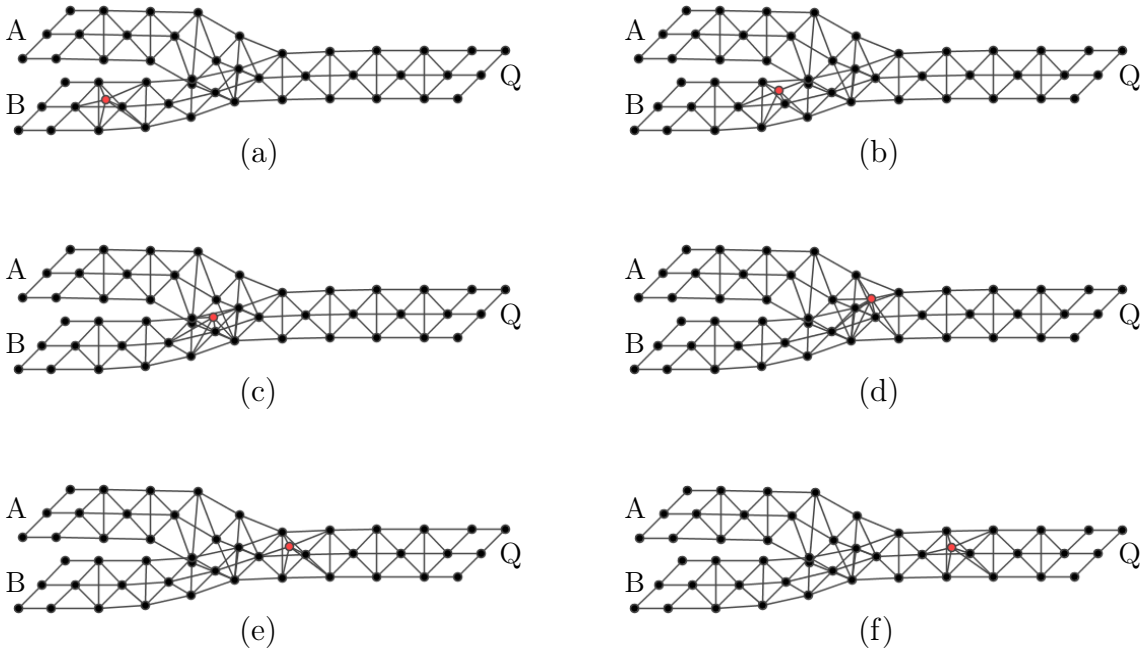


Figura 6.6: La gráfica G en (a) representa la configuración de la compuerta OR cuando en A entra un 0 y en B un 1. Las gráficas de (b) hasta (f) son $K^2(G)$ a $K^{10}(G)$, respectivamente. De (b) a (f) se muestra el traslado gradual del fotón hacia la salida Q.

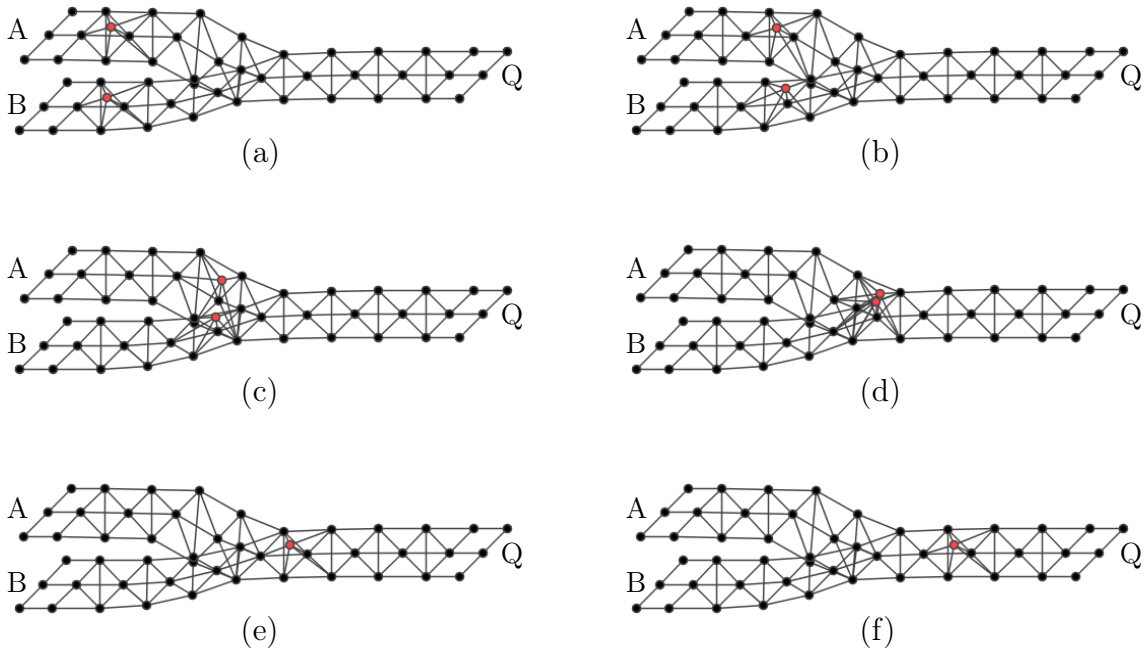


Figura 6.7: La gráfica G en (a) representa la configuración de la compuerta `OR` cuando en A y B entra un 1. Las gráficas de (b) hasta (f) son $K^2(G)$ a $K^{10}(G)$, respectivamente. Obsérvese que en las gráficas de (a) hasta (d) hay dos fotones, los cuales a partir de (e) se fusionan en uno solo para salir por Q.

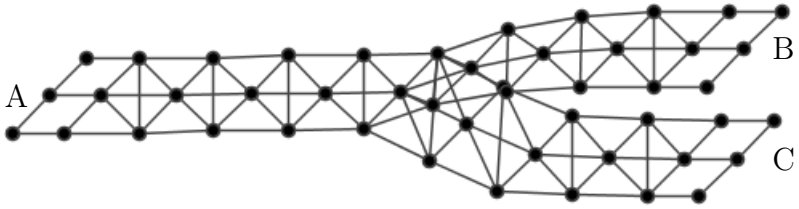


Figura 6.8: Divisor de señal con entrada en A y salidas en B y C. Nótese este componente es el reflejo horizontal de la gráfica que representa a la compuerta `OR`.

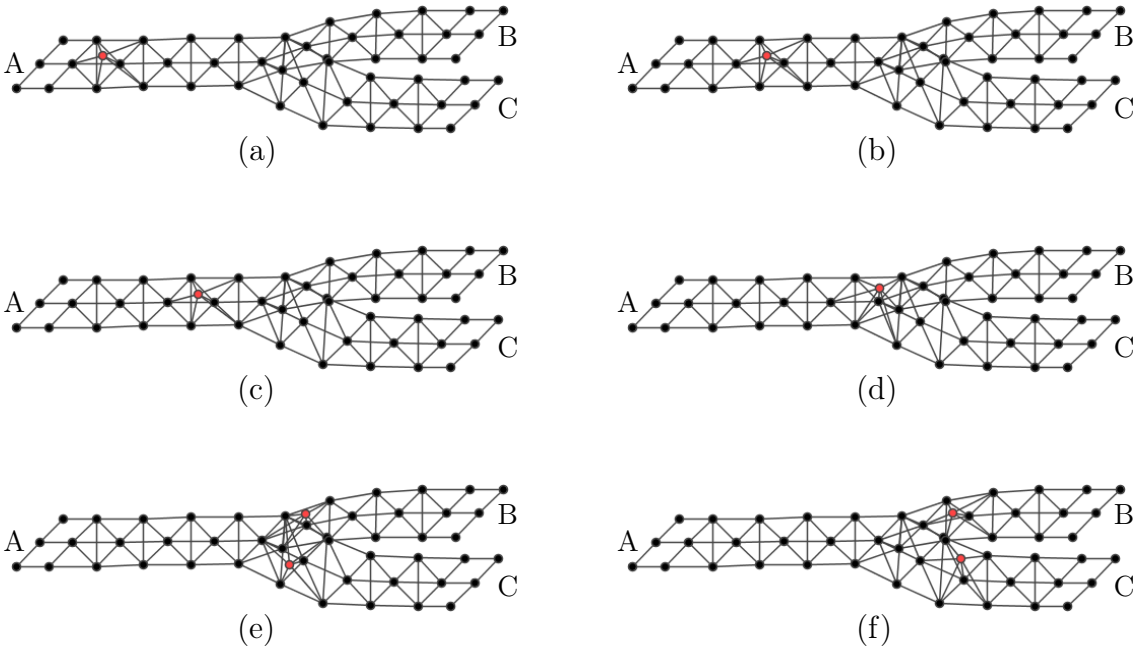


Figura 6.9: En (a) se muestra la gráfica G que representa al divisor de señal con un fotón en la entrada A. Las gráficas de (b) hasta (f) son $K^2(G)$ a $K^{10}(G)$, respectivamente. En la iteración $K^8(G)$, inciso (e), se observan dos fotones en canales distintos debido a que el operador de clanes duplica el fotón original. En las siguientes iteraciones, estos fotones continuarán trasladándose por los canales (ver inciso (f)) hasta llegar a las salidas B y C.

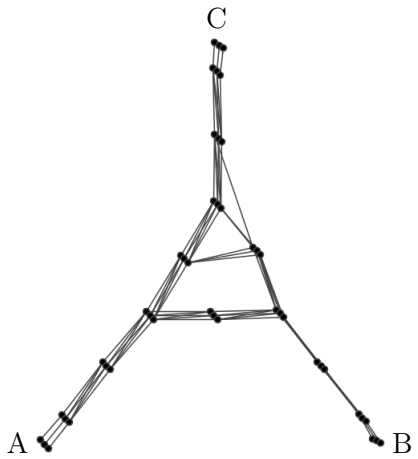


Figura 6.10: Empalme. Duplica el fotón entrante permitiendo que los dos fotones se distribuyan hacia los demás canales. Por ejemplo, si el fotón entra por A entonces saldrá un fotón por B y otro por C (ver figura 6.11).

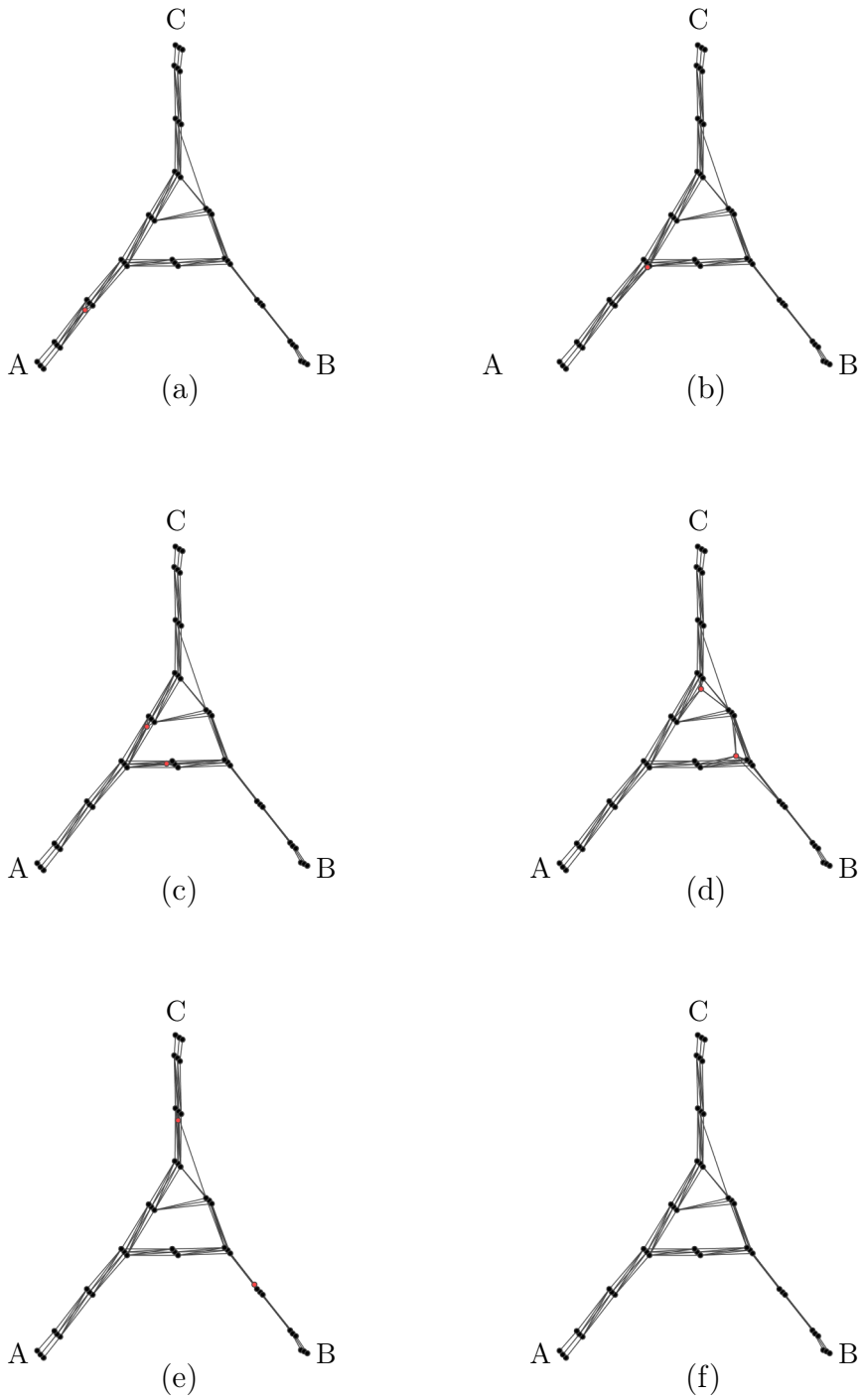


Figura 6.11: En (a) se muestra la gráfica G que representa al empalme con un fotón en la entrada A. Las gráficas de (b) hasta (f) son $K^2(G)$ a $K^{10}(G)$, respectivamente. En la iteración $K^4(G)$, inciso (e), se observa que el fotón de entrada en A es duplicado por el operador de clanes. En las siguientes dos iteraciones, $K^6(G)$ y $K^8(G)$, los fotones se trasladan hacia los canales B y C.

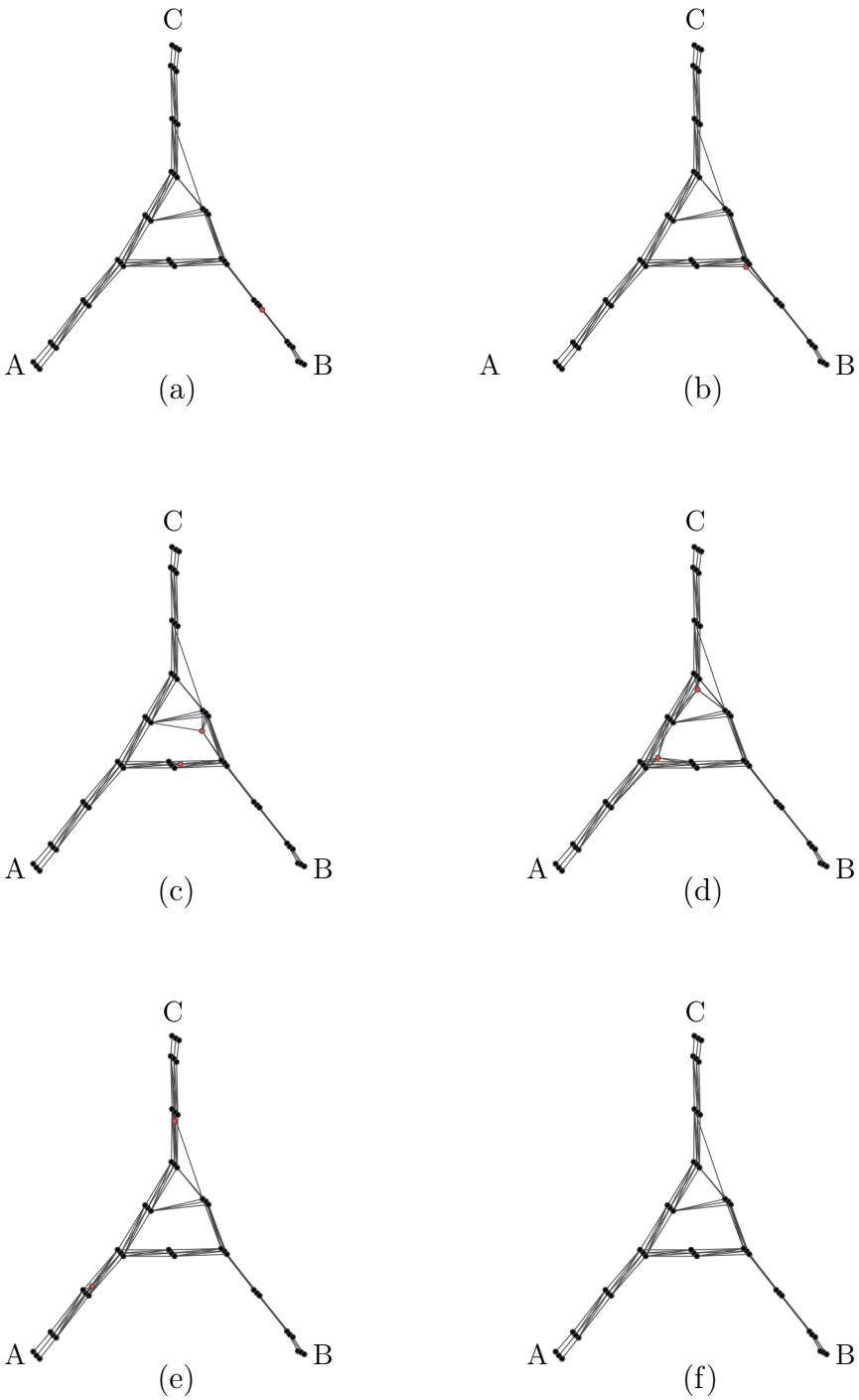


Figura 6.12: En (a) se muestra la gráfica G que representa al empalme con un fotón en la entrada B. Las gráficas de (b) hasta (f) son $K^2(G)$ a $K^{10}(G)$, respectivamente. En la iteración $K^4(G)$, inciso (e), se observa que el fotón de entrada en B es duplicado por el operador de clanes. En las siguientes dos iteraciones, $K^6(G)$ y $K^8(G)$, los fotones se trasladan hacia los canales A y C.

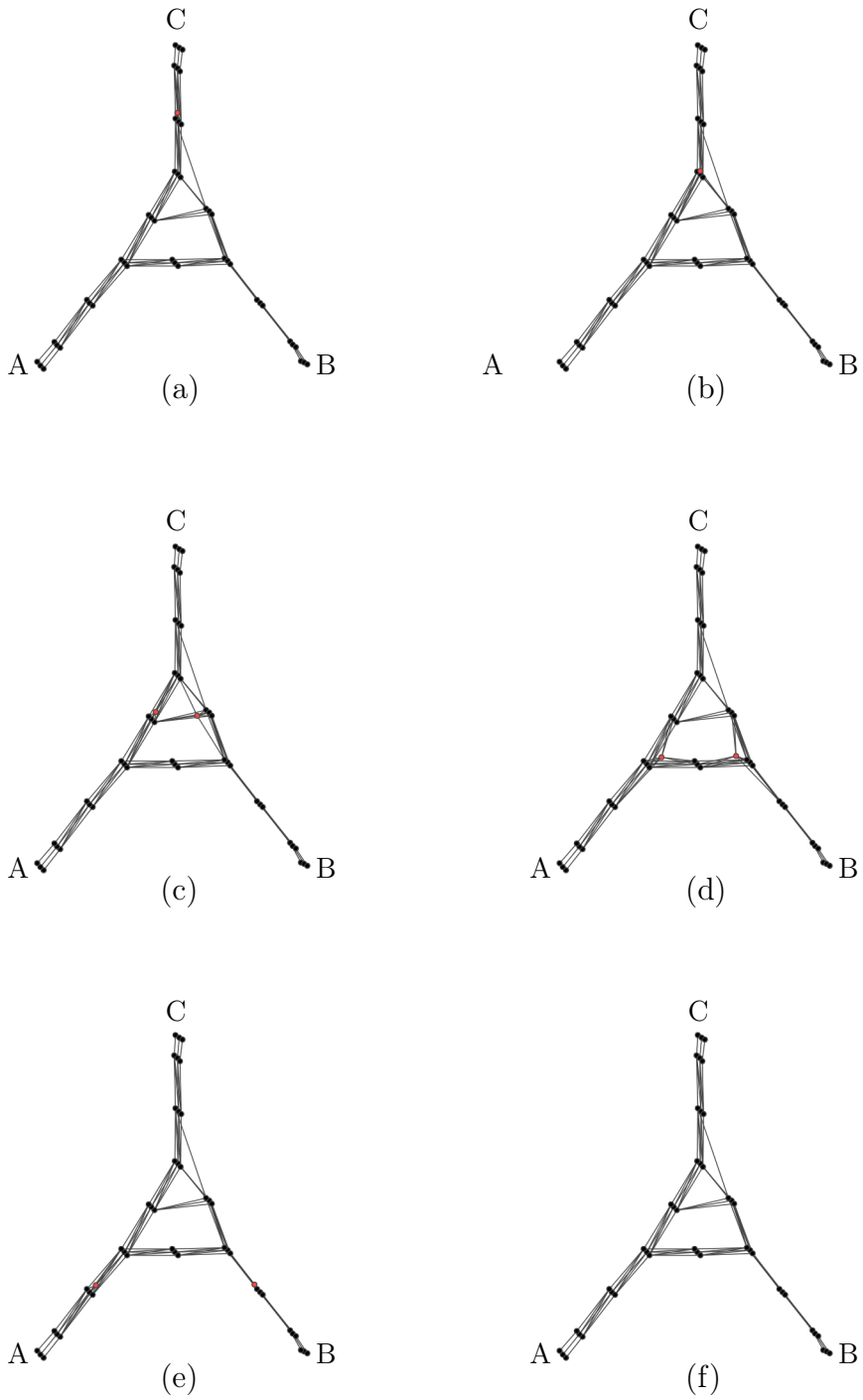


Figura 6.13: En (a) se muestra la gráfica G que representa al empalme con un fotón en la entrada C. Las gráficas de (b) hasta (f) son $K^2(G)$ a $K^{10}(G)$, respectivamente. En la iteración $K^4(G)$, inciso (e), se observa que el fotón de entrada en B es duplicado por el operador de clanes. En las siguientes dos iteraciones, $K^6(G)$ y $K^8(G)$, los fotones se trasladan hacia los canales A y B.

La afirmación del teorema 6.2.1 es un hecho computacional que es mejor verificarlo por medio de una computadora (por ejemplo, con el uso del software GAP + YAGS [9,22] y el código del apéndice A), aunque una explicación teórica también es posible: la principal consideración es que, salvo los fotones y la compuerta `AND`-sucia que se muestra en las figuras 6.14, 6.15, 6.16 y 6.17, todos los componentes son gráficas clan-Helly sin vértices dominados, y gracias a Escalante, se sabe que tales gráficas son K^2-invariantes. Entonces, el problema se reduce a analizar lo que pasa cerca de las perturbaciones locales (fotones) que se añaden.

El otro componente que se puede generar con gráficas de clanes es la compuerta `AND`-sucia. Ésta tiene el comportamiento de la compuerta `AND` (ver tabla 6.2) pero también genera muchos indeseables efectos secundarios (por eso mismo es llamada así). La simulación de cada fila, de arriba hacia abajo, de la tabla 6.2, es simulada respectivamente por las figuras 6.14, 6.15, 6.16 y 6.17. Por cuestiones técnicas, las gráficas que están en estas figuras tienen un canal extra que no está etiquetado. En particular, sólo la gráfica que se encuentra en el inciso (f) de la figura 6.17 presenta un vértice extra (pintado en color rojo) en la salida Q.

A	B	Q
0	0	0
0	1	0
1	0	0
1	1	1

Tabla 6.2: Definición de la compuerta `AND`.

De esta *primera codificación* la compuerta `AND`-sucia es la única que no es K^2-invariante debido a la cantidad de fotones que genera en todos los canales, por lo tanto no puede ser utilizada para simular circuitos digitales. Pese a esto, esta compuerta proporciona una pista para el desarrollo de la compuerta `AND` limpia, lo cual se conseguiría tratando de aislar el fenónemo que genera el vértice extra en la figura 6.17 (f) para así evitar los efectos secundarios.

Sin embargo, la tan necesitada compuerta `NOT` no es alcanzable como un componente *modular*. Por modular se referirá a que un componente es representado por una gráfica finita, que no es K-divergente, cuya estructura interna no depende de los alrededores del circuito. La condición de modularidad es necesaria para el teorema 6.2.2.

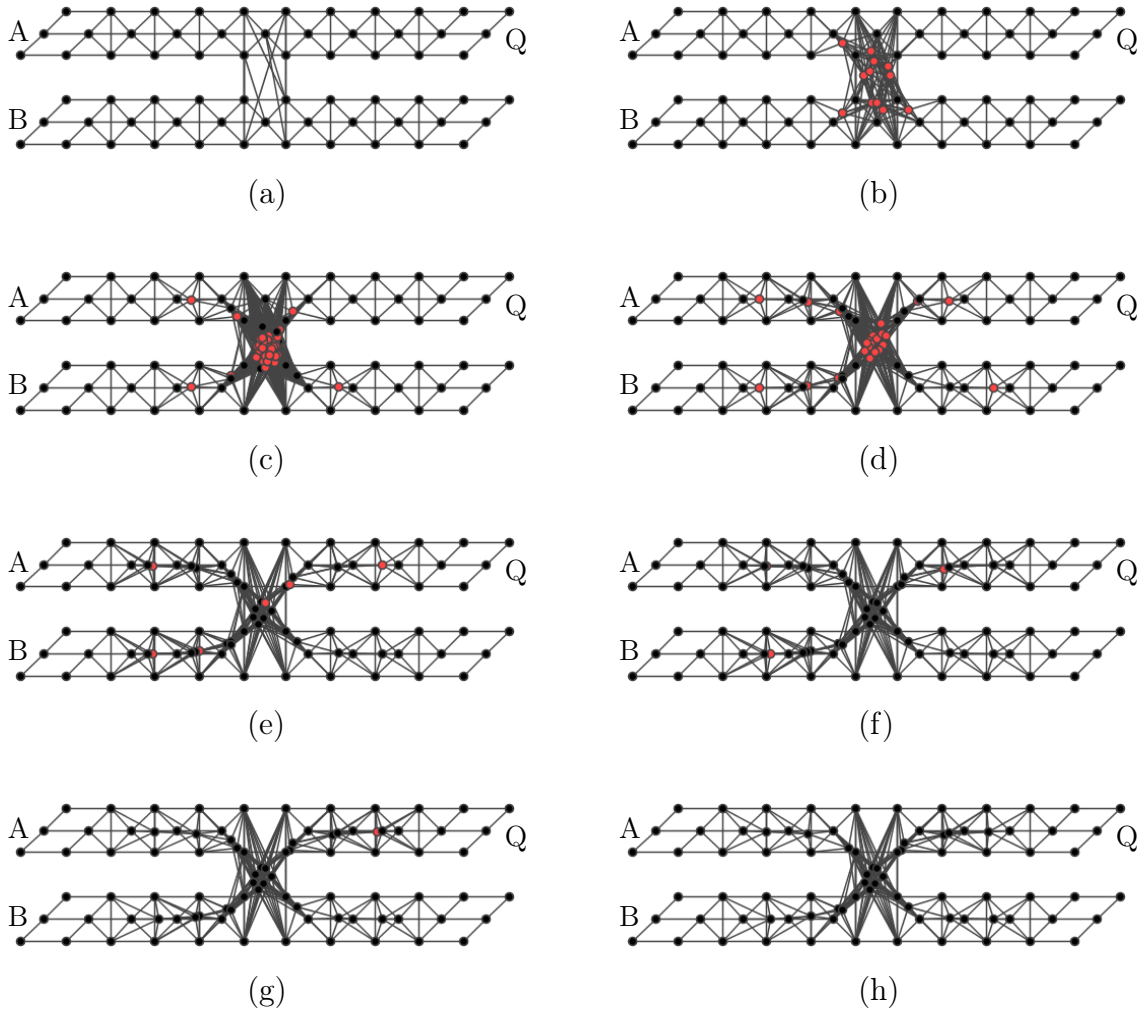


Figura 6.14: La gráfica G en (a) representa la configuración de la compuerta `AND` cuando en A y B entra un 0. Las gráficas de (b) hasta (h), que son $K^2(G)$ a $K^{14}(G)$ respectivamente, tienen una gran cantidad de fotones indeseados para simular el comportamiento de la compuerta `AND`. En particular el orden de la gráfica en (f) es de 93.

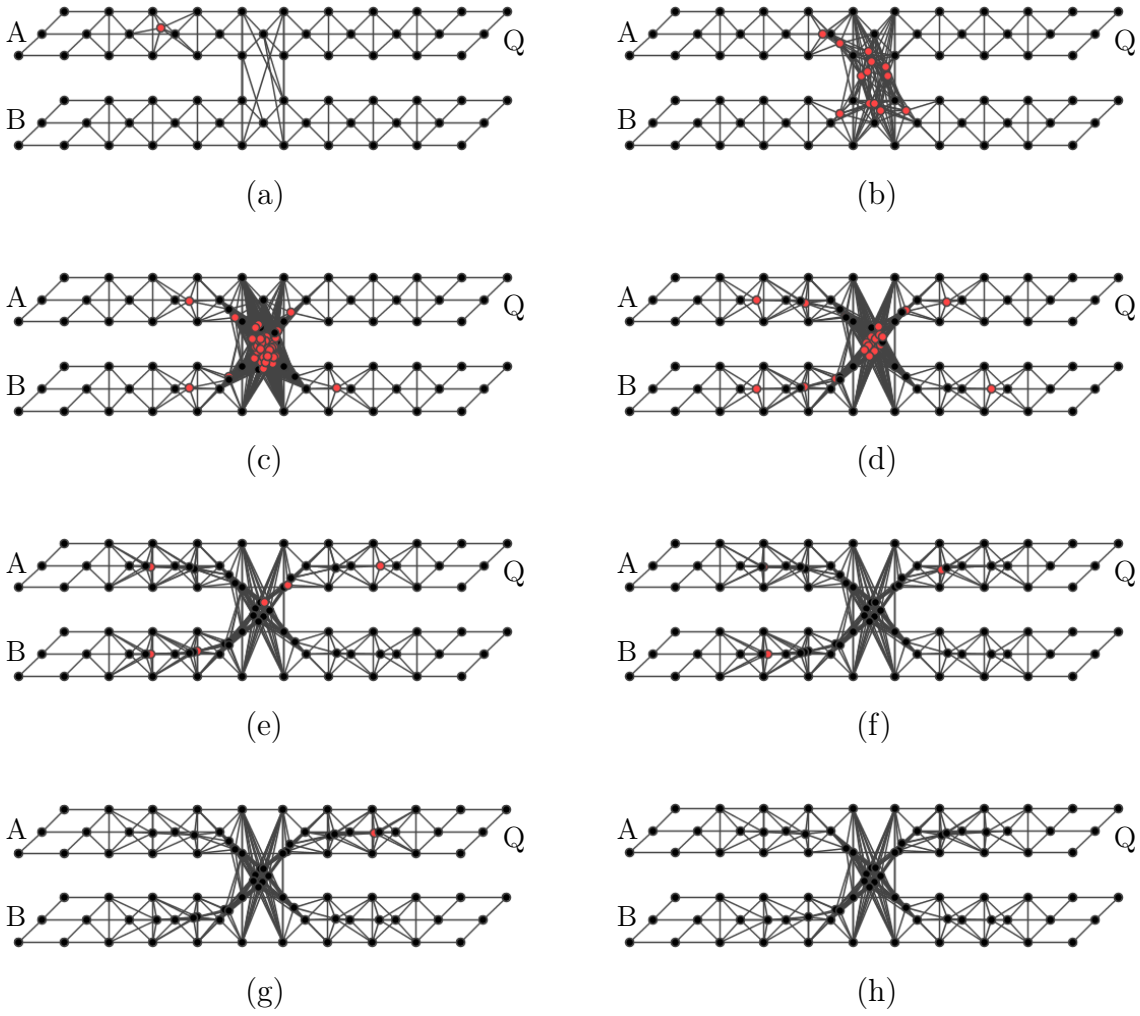


Figura 6.15: La gráfica G en (a) representa la configuración de la compuerta `AND` cuando en A entra un 1 y en B un 0. Las gráficas de (b) hasta (h) son $K^2(G)$ a $K^{14}(G)$, respectivamente. En particular el orden de la gráfica en (f) es de 93.

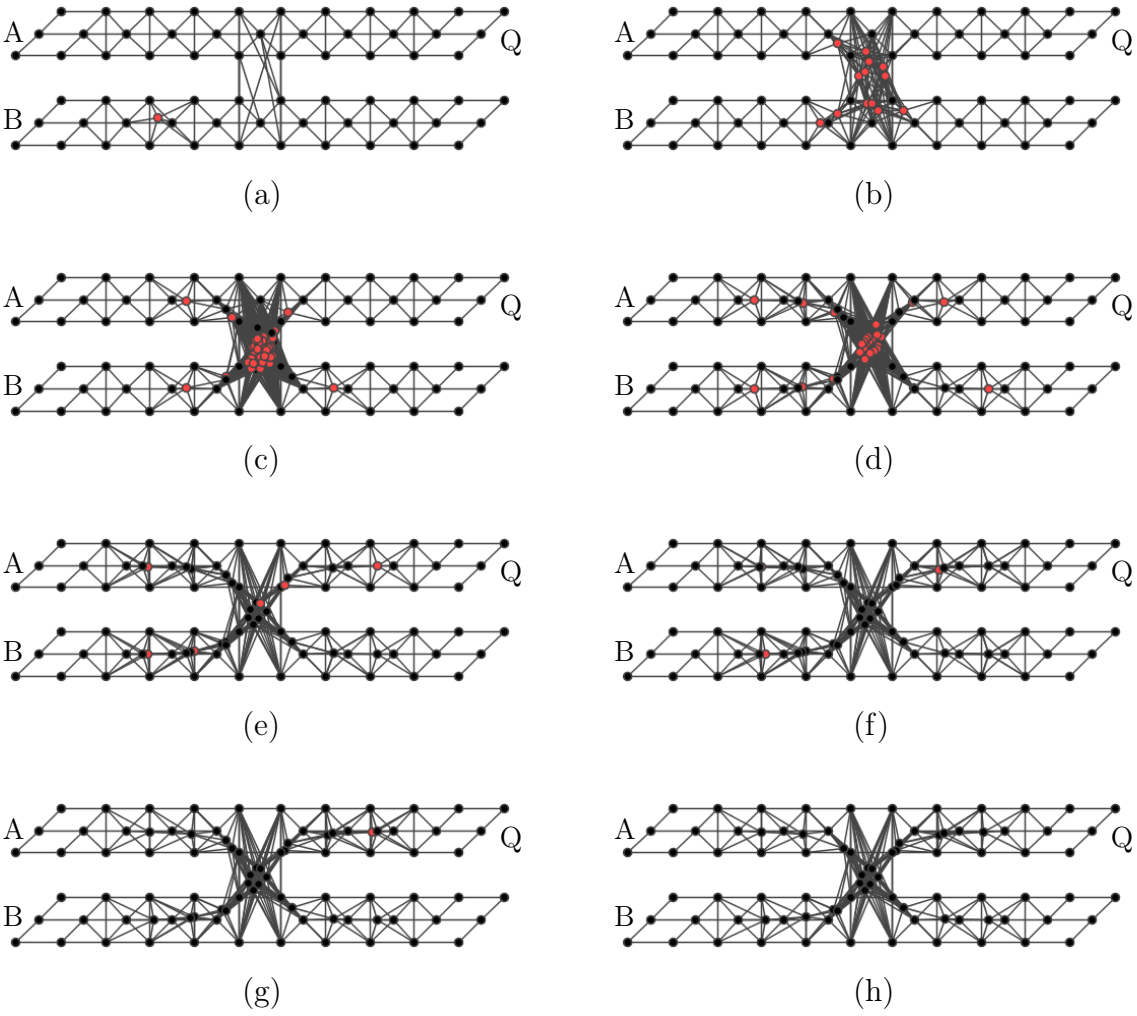


Figura 6.16: La gráfica G en (a) representa la configuración de la compuerta `AND` cuando en A entra un 0 y en B un 1. Las gráficas de (b) hasta (h) son $K^2(G)$ a $K^{10}(G)$, respectivamente. En particular el orden de la gráfica en (f) es de 94.

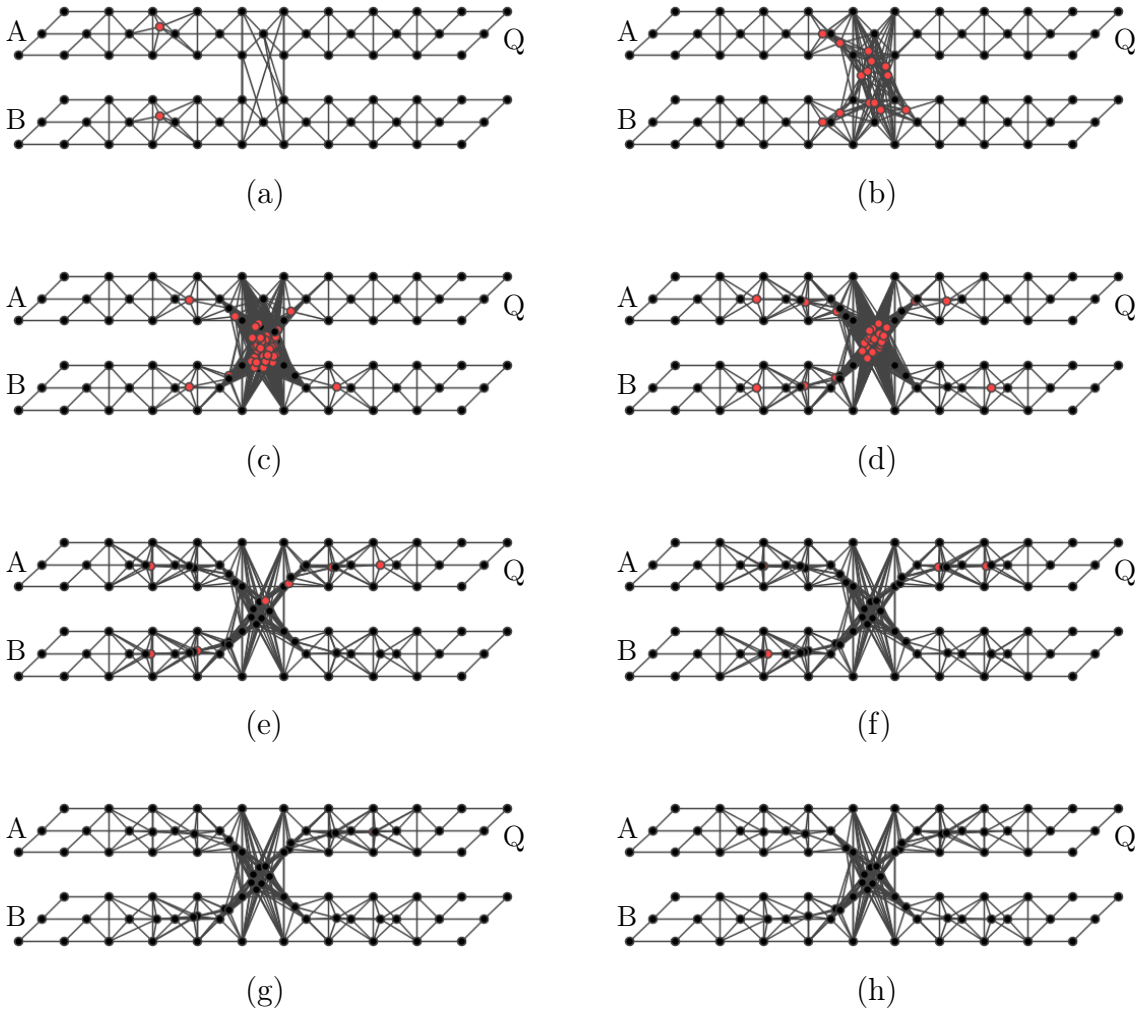


Figura 6.17: La gráfica G en (a) representa la configuración de la compuerta `AND` cuando en A y en B entra un 1. Las gráficas de (b) hasta (h) son $K^2(G)$ a $K^{10}(G)$, respectivamente. En el canal de salida Q de la gráfica (f) se está produciendo el fotón extra requerido por la función `AND`, por eso mismo, el orden de esta gráfica es de 95.

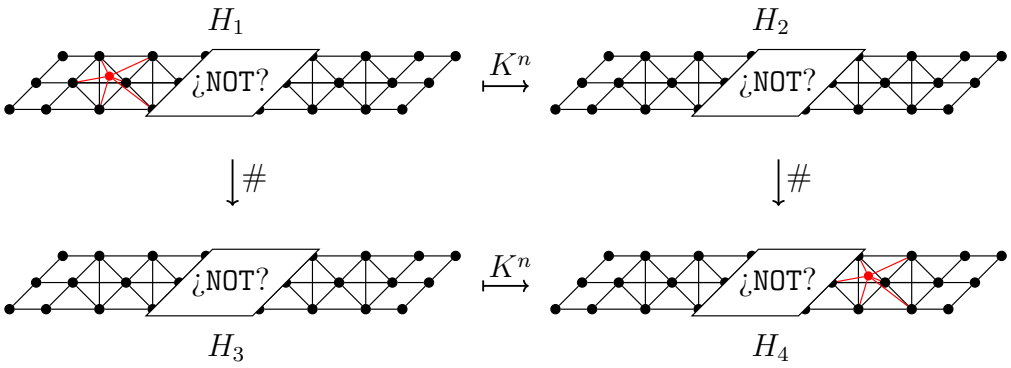


Figura 6.18: Esquema de la hipotética compuerta NOT. Éste consta de una caja negra que tiene conectado un canal de entrada y uno de salida. Las gráficas H_1 y H_3, que representan a la compuerta NOT cuando respectivamente entra 1 y un 0, después de n iteraciones del operador de clanes se transforman en H_2 y H_4, respectivamente.

Teorema 6.2.2. *Una compuerta modular* NOT *no puede ser simulada con gráficas de clanes usando la primera codificación.*

Demostración. Asúmase que se tiene la compuerta modular NOT, la cual se tratará como una caja negra que seguramente debe de tener un canal de entrada y uno de salida como se muestra en la figura 6.18.

Las gráficas H_1 y H_3 de la figura 6.18 representan a la compuerta NOT recibiendo en su entrada a un 1 y 0, respectivamente. Estas gráficas, después de un número finito de iteraciones del operador de clanes, deben ser transformadas en H_2 y H_4 de la figura 6.18, es decir $H_2 = K^n(H_1)$ y $H_4 = K^n(H_3)$. Por otra parte, el fotón en H_1 es un vértice dominado en la misma gráfica y por lo tanto (ya que la compuerta es modular) H_1 es desmantelable a H_3 en un sólo paso, es decir $H_1 \xrightarrow{\#} H_3$.

Por el teorema 2.2.3, se sabe que $H_2 = K^n(H_1) \xrightarrow{\#} K^n(H_3) = H_4$, la cual, por definición, implica que H_2 contiene a una subgráfica H_4' isomorfa a H_4.

Si primero se supone que el isomorfismo de H_4' a H_4 envía el canal de salida de H_2 a el canal de salida de H_4; entonces se tiene una contradicción ya que no hay ningún vértice de H_4' que podría ser mapeado a el fotón saliente de H_4.

Ahora supóngase lo contrario, que el isomorfismo entre H_4' y H_4 no preserva el canal de salida. Ésto en principio podría pasar si algún fotón en la estructura interna de la compuerta NOT, en H_4', es mapeado en el fotón saliente de H_4. Sin embargo, la compuerta NOT, por hipótesis, es modular y por lo tanto su estructura interna no depende de los alrededores del circuito. Como se puede conectar cualquier circuito al canal de salida de la compuerta NOT (por ejemplo, un canal de salida de cualquier longitud deseada) y además la compuerta NOT debe producir el isomorfismo entre H_4' y H_4 en todos y cada uno de los casos, entonces, la compuerta NOT o bien debe (1) *contener* una copia interna preexistente de todos los circuitos que se puedan conectar

al canal de salida (lo cual contradice la suposición de finitud de la modularidad) o bien debe (2) *construir* una copia interna de todos los circuitos que se puedan conectar al canal de salida (lo cual contradice la suposición de la no clan-divergencia de la modularidad). De esto se sigue que el isomorfismo entre H_4' y H_4 debe preservar el canal de salida como en el párrafo anterior y por lo tanto la compuerta `NOT` modular no existe. □

6.3. Segunda codificación

En vista del teorema 6.2.2, aquí se plantea una nueva codificación que reutiliza los componentes de la primera codificación para poder transportar información (ceros y unos) a través de *canales dobles*, formados por un canal superior y un canal inferior (ver figura 6.19).

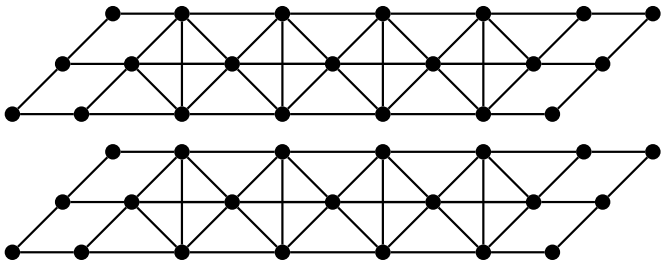
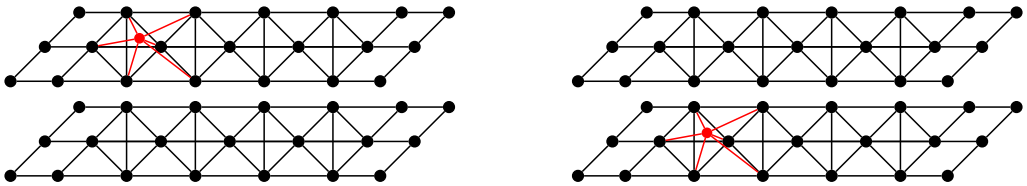


Figura 6.19: Canal doble.



(a) (b)

Figura 6.20: Segunda codificación. En (a) y (b) se muestra la codificación respectivamente del 1 y del 0 usando canales dobles.

Segunda codificación: El símbolo 1 se codifica colocando un fotón en el canal superior y ninguno en el canal inferior; mientras que símbolo 0 se codifica de manera

contraria, es decir, colocando un fotón en el canal inferior y ninguno en el canal superior. Las otras dos posibles combinaciones son consideradas inválidas.

Con esta segunda codificación es posible simular a la compuerta NOT con tan sólo intercambiar los canales como se muestra en la figura 6.21, donde cada línea representa un canal.

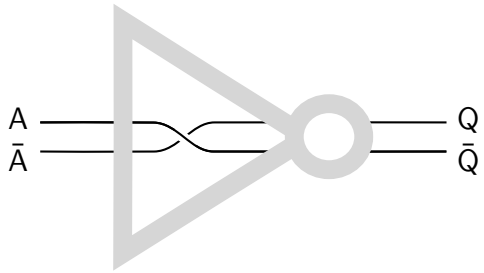


Figura 6.21: Compuerta NOT. Las líneas etiquetadas con A y $\bar{A}$ representan respectivamente al canal superior e inferior del canal doble de entrada en la compuerta NOT; mientras que las líneas Q y $\bar{Q}$ representan al canal superior e inferior del canal doble de salida.

Muchos de los componentes vistos en la *primera codificación* pueden volver a implementarse fácilmente en esta codificación tan sólo haciendo copias de ellos, es decir, haciendo una copia para el canal superior y otra copia para el canal inferior. Por lo cual se plantea el siguiente teorema:

Teorema 6.3.1. *Usando gráficas de clanes y la segunda codificación se puede simular canales, portadores de señal, divisores de señal, empalmes y compuertas* NOT.

Los siguientes resultados dependen de la existencia de la compuerta AND-limpia de la primera codificación (recuérdese que hasta el momento sólo se cuenta con la compuerta AND-sucia) ya que con ella y con la compuerta OR se diseña a la compuerta AND en esta segunda codificación (ver figura 6.22). Esto se establece en el siguiente teorema:

Teorema 6.3.2. *Si la compuerta* AND *puede ser simulada mediante gráficas de clanes en la primera codificación, entonces la compuerta* AND *también puede ser simulada en la segunda codificación.*

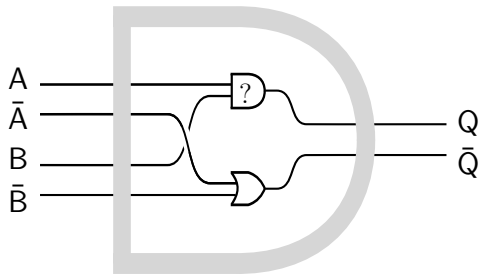


Figura 6.22: Compuerta AND. Está formada por la compuerta OR y AND de la primera codificación. La compuerta AND tiene el símbolo "**?**" ya que se está considerando la posible existencia de la compuerta AND-limpia de la primera codificación.

Con un conjunto funcionalmente completo de compuertas (en este caso $\{\texttt{AND}, \texttt{NOT}\}$), si se puede usar gráficas infinitas, se puede simular cualquier máquina de Turing. No se necesita una clase enorme de gráficas infinitas para esta simulación (como la clase de gráficas numerables que tiene cardinalidad del continuo), es suficiente usar *gráficas cuasi periódicas*, es decir, gráficas finitas que son repetidas un número numerable de veces donde algunos vértices en la i-ésima copia pueden ser identificados con algunos vértices en la $(i+1)$-ésima copia, en una manera periódica, con solo un número finito de vértices y aristas adicionales. Las gráficas cuasi-periódicas pueden ser finitamente representadas y por lo tanto la clase formada por ellas es numerable.

Teorema 6.3.3. *Si la compuerta* AND *puede ser simulada en la primera codificación, entonces el operador de las gráficas de clanes es Turing-completo para gráficas (infinitas pero) cuasi-periódicas.*

Cabe mencionar que el teorema 6.3.3 depende de la simulación de las máquinas de Turing que se muestra en el capítulo 7. Además, si se hace la restricción a gráficas finitas y se asume que existe la compuerta AND de la primera codificación, entonces debe ser claro que la mismas técnicas pueden ser usadas para simular una computadora de memoria finita (es decir, una computadora real) usando solo gráficas finitas. En particular, un Autómata Linealmente Acotado (LBA), ver sección 2.6, puede ser simulado usando gráficas de clanes finitas si la primera codificación de la compuerta AND existe.

Capítulo 7

Simulación de las máquinas de Turing

En este capítulo se muestra cómo se puede simular una máquina de Turing usando (infinitas) compuertas lógicas. Esto ya se había esbozado en la literatura en [14], pero aquí se procederá con una simulación más explícita donde se contempla que las compuertas lógicas están simuladas con gráficas de clanes (como se muestra en el capítulo anterior).

Las máquinas de Turing que se van a simular son *estrictamente binarias*. Estas se caracterizan porque su alfabeto es $\{0, 1\}$ en donde se descarta al símbolo del espacio en blanco ($\textvisiblespace$). Es fácil mostrar que las máquinas de Turing estrictamente binarias pueden simular a cualquier máquina de Turing estándar (cuyo alfabeto contiene cualquier conjunto de símbolos más el espacio en blanco) empleando las técnicas que se usan en la demostración del teorema 7.10 en [27].

La simulación general contempla la hipótesis de la existencia de la compuerta `AND`-limpia en la primera codificación, ya que con ella, se tendría a la compuerta `AND` en la segunda codificación (ver figura 6.22). Al tener esta última se obtendría al conjunto funcionalmente completo de compuertas lógicas $\{\texttt{AND}, \texttt{NOT}\}$ y por lo tanto, se podría diseñar a la compuerta `OR` y a otros componentes más complejos (como los *flip-flops* tipo SR o D que permiten almacenar información en el circuito).

En conjunto, las compuertas lógicas, los cables y los *flip-flops* permiten la construcción del circuito digital que simula a la unidad de control y a las celdas de memoria de la máquina de Turing (ver figuras 7.5, 7.6 y 7.7).

El diseño global de la simulación se detalla en la figura 7.1, donde a la derecha de la unidad de control están las celdas de memoria representadas con rectángulos. El círculo en la parte superior de cada celda sirve para indicar la posición de la cabeza, entonces, sólo si éste está relleno, significa que la cabeza está posicionada en esa localidad de memoria.

La cinta de la figura 7.1 contiene una celda especial llamada *celda cero*, localizada a lado de la unidad de control, que detecta si la cabeza se ha desbordado a la izquierda de la cinta. Esta celda de memoria no almacena ningún dato. El resto de las celdas, llamadas *i-ésimas celdas* tal que $i > 0$, contienen un círculo lleno o

vacío para almacenar respectivamente la presencia o ausencia de la cabeza; además almacenan los símbolos (0 o 1) pertenecientes a la cadena de entrada w de la máquina de Turing. Las celdas que están a la derecha de aquella que tiene el último símbolo de w deben de almacenar un cero. Por ejemplo, en la figura 7.1 está almacenada la cadena $w = 10111$ y posteriormente las celdas siguientes contienen ceros.

Las señales que se envían de la unidad de control hacia las celdas de memoria pasan a través del *bus de salida*, mientras que las señales que van de las celdas de memoria hacia la unidad de control pasan a través del *bus de entrada*. Los canales (representados por flechas) que hay entre una celda y otra transportan a las señales locales $\texttt{HF}_\texttt{i}$ y $\texttt{HB}_\texttt{i}$, las cuales sirven para enviar la señal de activación de la cabeza hacia la celda posterior y anterior, respectivamente.

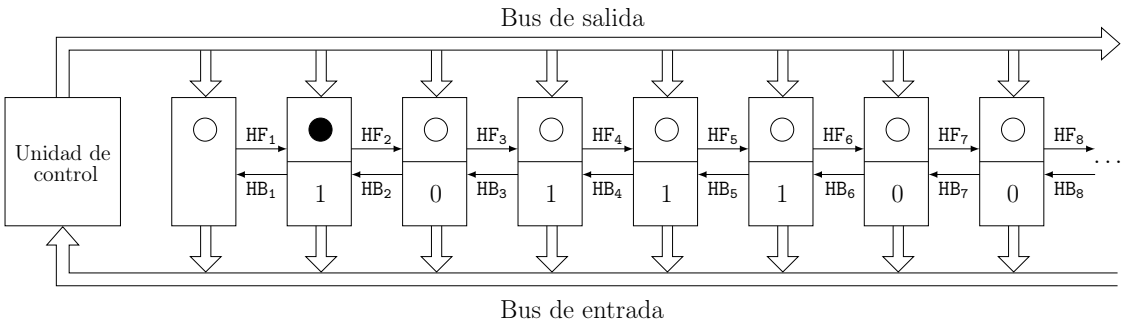


Figura 7.1: Esquema del diseño digital que simula la máquina de Turing. Los rectángulos alargados representan a las celdas de memoria y el círculo relleno marca la presencia de la cabeza en la celda correspondiente (de lo contrario, el círculo vacío significa que la cabeza no está en ella). La celda cero (localiza a lado de la unidad de control) no almacena ningún dato.

La simulación propuesta en este capítulo da pie al siguiente teorema:

Teorema 7.0.1. *Cualquier máquina de Turing puede ser simulada con un circuito digital infinito, pero cuasi-periódico.*

7.1. Flip-flops

En el capítulo anterior se mencionó que los *flip-flops* son circuitos para el almacenamiento de bits en el diseño de una computadora digital. Estos circuitos son secuenciales ya que sus salidas dependen no sólo de las entradas actuales, sino también de la historia de las entradas anteriores, equivalentemente, sus salidas dependen de las entradas y del contenido que tengan en los elementos de memoria. Un *flip-flop* mantiene su estado binario indefinidamente hasta que llega una señal que le indica que debe de cambiar de estado.

Uno de los *flip-flops* más básicos que se puede construir es el SR. Éste está formado por dos compuertas `NAND` acopladas en cruz (ver figura 7.2) y por dos entradas `S` (*set*, establecer) y `R` (*reset*, restablecer), las cuales se encuentran negadas. Además tiene dos salidas `Q` y $\overline{\texttt{Q}}$ que son complemento una de otra excepto cuando en ambas entradas hay un 1, ya que en las dos salidas se genera un 1 (en este caso se dice que se presenta un estado indefinido y se representa por el símbolo "?").

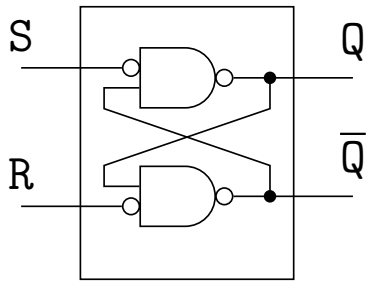


Figura 7.2: *Flip-flop* SR.

S	R	Q(t + 1)
0	0	Q(t)
0	1	0
1	0	1
1	1	?

Tabla 7.1: Definición del *flip-flop* SR. El símbolo "?" indica estado indefinido.

La tabla 7.1 muestra que cuando `S` = 1 y `R` = 0 se *ajusta* el contenido de las salidas `Q` y $\overline{\texttt{Q}}$. Cuando `S` regresa a 0 se mantiene el contenido de las salidas, como si fuese una especie de *memoria* del paso anterior. De manera análoga, en el momento en que ambas entradas regresen a 0, se puede cambiar `R` = 1, lo cual ajustará nuevamente las salidas y posteriormente, cuando `R` regrese a 0, se mantendrá la memoria de las salidas anteriores. El caso `S` = `1` y `R` = `1` se considera indefinido en el *flip-flop* SR y en general se desea evitar.

Otro tipo de *flip-flop* que se usa es el *flip-flop* D (ver figura 7.3). Nótese que este *flip-flop* D consta de dos entradas, `D` y `CLK`, y de dos salidas, `Q` y $\overline{\texttt{Q}}$; por lo que `S` y `R` sólo están para indicar que a partir de allí hacia la derecha se tiene un *flip-flop* SR. Entonces, la entrada `D` pasa directamente a `S` y su complemento pasa a `R` cuando la señal del reloj, `CLK`, así lo indique.

Mientras la entrada `CLK` (que es el pulso del reloj) se encuentre en 0, llegarán a `S` y a `R` el valor de 0, por lo que no habrá cambio alguno en las salidas independientemente del valor que tome `D`. Cuando `CLK` este activo (es decir, con valor 1) y `D` = 0, a `S` y

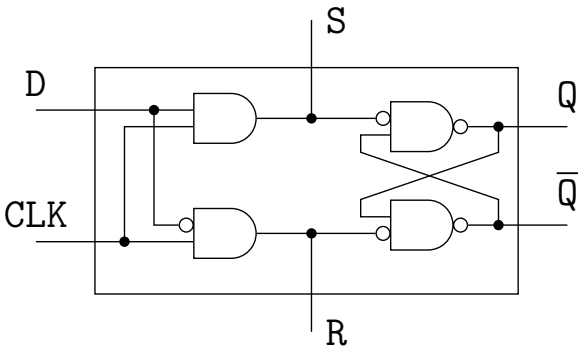


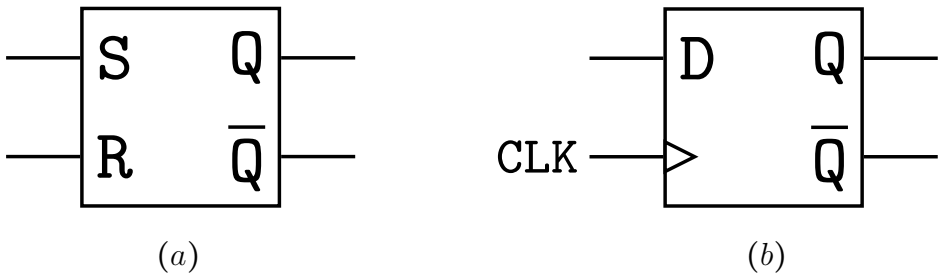
Figura 7.3: *Flip-flop* D.

CLK	D	Q(t + 1)
0	X	Q(t)
1	0	0
1	1	1

Tabla 7.2: Definición del *flip-flop* D.

a R llegará un 0 y un 1, respectivamente, por lo que la salida Q(t + 1) tendrá un 0 (ver tabla 7.2). De manera análoga, cuando CLK = 1 y D = 1, a S y a R llegará un 1 y un 0, respectivamente, por lo que la salida Q(t + 1) tendrá un 1. Nótese que dado este comportamiento, el *flip-flop* D transporta el dato que hay en D hacia la salida Q siempre y cuando CLK = 1.

En lo que resta del capítulo se usarán los símbolos gráficos que se muestran en la figura 7.4 para representar al *flip-flop* SR y al *flip-flop* D. En ambas representaciones las entradas están del lado izquierdo y las salidas del lado derecho. La salida $\overline{Q}$ se omitirá lo que resta del capítulo para ambos flip-flops.



(*a*) (*b*)

Figura 7.4: (*a*) Símbolo gráfico del *flip-flop* SR. (*b*) Símbolo gráfico del *flip-flop* D.

7.2. Unidad de control

En esta sección se presenta cómo crear la unidad de control (UC) de cualquier máquina de Turing. Para ilustrar esto, se usará un ejemplo en partícular de una MT, cuya función de transición δ y unidad de control se muestran respectivamente en la tabla 7.3 y la figura 7.5.

	0	1
q_0	$(q_1, 0, \rightarrow)$	$(q_0, 1, \rightarrow)$
q_1	$(q_1, 1, \leftarrow)$	$(q_2, 0, \leftarrow)$
q_2	$(q_2, 1, \rightarrow)$	–

Tabla 7.3: Función de transición δ de una máquina de Turing con el conjunto de estados $\{q_0, q_1, q_2\}$, donde q_0 es el estado inicial y q_2 es el estado final, con alfabetos $\Sigma = \{0, 1\}$ y $\Gamma = \{0, 1\}$.

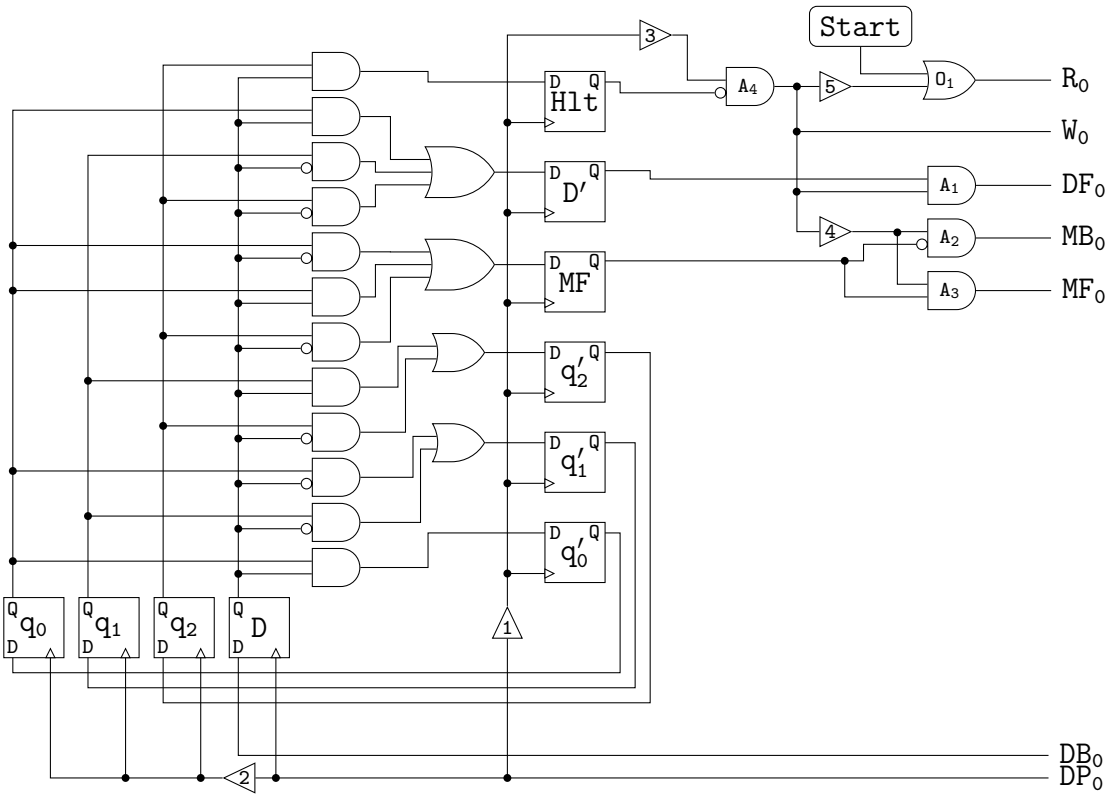


Figura 7.5: Diseño de la unidad de control (UC) de la máquina de Turing definida en la tabla 7.3.

De manera general, toda unidad de control contiene dos conjuntos de *flip-flops* tipo D: 1) *estatus actual* y 2) *estatus nuevo*, los cuales respectivamente, tienen $n = |Q| + 1$ y $m = |Q| + 3$ *flip-flops*, donde $|Q|$ es la cantidad de estados de la máquina de Turing en cuestión.

Los *flip-flops* del conjunto *estado actual* están etiquetados mediante $\mathtt{q_i}$ ($0 \leq \mathtt{i} \leq |Q|$) y $\mathtt{D}$. Sólo en uno de los $\mathtt{q_i}$ se almacena un 1 en caso de que $\mathtt{q_i}$ sea el estado actual de la MT, mientras que en el resto de los $\mathtt{q_i}$ se guarda un 0. En $\mathtt{D}$ se almacena el dato actual al que la cabeza está apuntado en la cinta. Es por esto, que en la figura 7.5, este conjunto contiene cuatro *flip-flops*: $\mathtt{q_0}, \mathtt{q_1}, \mathtt{q_2}$ y $\mathtt{D}$.

Los *flip-flops* del conjunto *estado nuevo* están etiquetados mediante $\mathtt{q'_i}$ ($0 \leq \mathtt{i} \leq |Q|$), $\mathtt{D'}$, $\mathtt{MF}$ y $\mathtt{Hlt}$. En un solo $\mathtt{q'_i}$ se almacena un 1 en caso de que $\mathtt{q_i}$ sea el nuevo estado al que pasará la MT, mientras que en el resto se guarda un 0. En $\mathtt{D'}$, $\mathtt{MF}$ y $\mathtt{Hlt}$ se almacena respectivamente, el dato que se escribirá en la cinta, el movimiento de la cabeza (1 para la derecha y 0 para la izquierda) y el paro de la máquina de Turing (0 para continuar y 1 para detenerse). Es por esto, que en la figura 7.5, este conjunto contiene seis *flip-flops*: $\mathtt{q'_0}$, $\mathtt{q'_1}$, $\mathtt{q'_2}$, $\mathtt{D'}$, $\mathtt{MF}$ y $\mathtt{Hlt}$.

El valor almacenado en cada *flip-flop* del conjunto estatus nuevo depende del resultado de su *fórmula lógica*, etiquetada igual que su respectivo *flip-flop*, la cual está en forma normal disyuntiva (FND) y usa como términos a los estados ($\mathtt{q_i}$) y al dato leído en la cinta $\mathtt{D}$, o su negación $\overline{\mathtt{D}}$. Estas fórmulas lógicas siempre pueden ser generadas a través de la función de transición de una máquina de Turing, para posteriormente ser representadas mediante compuertas lógicas. Por ejemplo, el conjunto de fórmulas para la máquina de Turing definida en la tabla 7.3 son las siguientes:

1. $\mathtt{q'_0} = \mathtt{q_0} \wedge \mathtt{D}$
2. $\mathtt{q'_1} = (\mathtt{q_0} \wedge \overline{\mathtt{D}}) \vee (\mathtt{q_1} \wedge \overline{\mathtt{D}})$
3. $\mathtt{q'_2} = (\mathtt{q_1} \wedge \mathtt{D}) \vee (\mathtt{q_2} \wedge \overline{\mathtt{D}})$
4. $\mathtt{D'} = (\mathtt{q_0} \wedge \mathtt{D}) \vee (\mathtt{q_1} \wedge \overline{\mathtt{D}}) \vee (\mathtt{q_2} \wedge \overline{\mathtt{D}})$
5. $\mathtt{MF} = (\mathtt{q_0} \wedge \overline{\mathtt{D}}) \vee (\mathtt{q_0} \wedge \mathtt{D}) \vee (\mathtt{q_2} \wedge \overline{\mathtt{D}})$
6. $\mathtt{Hlt} = (\mathtt{q_2} \wedge \mathtt{D})$

Para generar las fórmulas $\mathtt{q'_i}$ se toman las condiciones en las que debe de estar la máquina de Turing en los casos donde se transita al estado $\mathtt{q_i}$, tal como lo indica la función de transición δ. Por esta razón, para $\mathtt{q'_1}$ se consideran los dos siguientes casos (ver tabla 7.3):

1. $(\mathtt{q_0} \wedge \overline{\mathtt{D}})$: cuando se está en el estado $\mathtt{q_0}$ y se lee el dato 0, o
2. $(\mathtt{q_1} \wedge \overline{\mathtt{D}})$: cuando se está en el estado $\mathtt{q_1}$ y se lee el dato 0.

De manera similar a como se obtiene cada $\mathtt{q}'_\mathtt{i}$, la fórmula lógica `Hlt` sólo considera los casos donde se detiene la máquina de Turing. Para el ejemplo considerado esto sucede cuando se está en el estado $\mathtt{q}_2$ y se lee un 1 en la cinta: $(\mathtt{q}_2 \wedge \mathtt{D})$.

Mientras que para las fórmulas `MF` y $\mathtt{D}'$ sólo se consideran las condiciones de los casos donde éstas toman valor 1, es decir, cuando respectivamente la función de transición indique un movimiento a la derecha y la escritura de un 1 en la cinta.

Entonces, para cualquier máquina de Turing, una vez que ya fueron generadas las fórmulas lógicas (a partir de su función de transición) se obtiene la representación mediante compuertas lógicas, donde por cada conjunción ($\wedge$) se coloca una compuerta `AND` y por cada disyunción ($\vee$) una compuerta `OR`. En la figura 7.5, las fórmulas lógicas de la máquina de Turing que se está tomando como ejemplo (definida en la tabla 7.3) están representadas mediante el conjunto de compuertas `AND` y `OR` que están sin etiquetar a la izquierda de la figura.

La unidad de control y las celdas de memoria tienen señales que cuando están activas transportan información en forma de *pulsos* (ya que pasan de manera transitoria de un 0 a un 1), de lo contrario, cuando están inactivas transportan ceros hasta que vuelvan a activarse. Se consideran tres tipos de señales:

1. Señales de control: Son aquellas que indican la ejecución de cierta instrucción desde la unidad de control hacia las celdas de memoria o viceversa.

2. Señales locales: Son aquellas que indican la ejecución de cierta instrucción desde la i-ésima hacia la $\{i+1\}$-ésima celda de memoria o viceversa.

3. Señal de dato: Son aquellas que transportan el dato leído o el dato a escribir en la cinta. Estas van desde la i-ésima celda hacia la unidad de control y viceversa.

La unidad de control, bajo esta simulación, de cualquier máquina de Turing tiene cinco señales de control: $\mathtt{R}_0$, $\mathtt{W}_0$, $\mathtt{MB}_0$, $\mathtt{MF}_0$ y $\mathtt{DP}_0$; y dos señales de dato: $\mathtt{DF}_0$ y $\mathtt{DB}_0$. A continuación la descripción de cada una de ellas:

1. $\mathtt{DB}_0$ (Datum Backward): Dato que viene del que está almacenado en la celda memoria a la que está apuntando la cabeza de la máquina de Turing.

2. $\mathtt{DP}_0$ (Datum Present): Indica que se está enviando el dato de la celda de memoria hacia unidad de control en ese momento.

3. $\mathtt{R}_0$ (Read): Solicitud de la unidad de control para que se lea el dato almacenado en la celda donde la cabeza de la máquina de Turing está apuntando.

4. $\mathtt{W}_0$ (Write): Solicitud de la unidad de control para que se escriba un dato en la celda donde la cabeza de la máquina de Turing está apuntado.

5. $\mathtt{DF}_0$ (Datum Forward): Dato que se envía para ser escrito en la celda de memoria a la que está apuntando la cabeza de la máquina de Turing.

6. $\texttt{MB}_0$ (Move Backward): Indica que la cabeza de la máquina de Turing tendrá que hacer un movimiento a la celda de la izquierda.

7. $\texttt{MF}_0$ (Move Forward): Indica que la cabeza de la máquina de Turing tendrá que hacer un movimiento a celda de la derecha.

La descripción de las señales en toda la máquina de Turing es la misma independientemente del subíndice que tengan. Este último sólo ayuda a identificar el orden de conexión entre las celdas de memoria. Por ejemplo, para este caso el subíndice 0 indica que las señales están conectadas a la celda cero.

En toda unidad de control se considera que $\texttt{R}_0$, $\texttt{W}_0$, $\texttt{DF}_0$, $\texttt{MB}_0$ y $\texttt{MF}_0$ son señales de salida; mientras que $\texttt{DB}_0$ y $\texttt{DP}_0$ son señales de entrada. Además de estas, también existe la señal `Start`, la cual inicia el proceso de ejecución en la máquina de Turing cuando recibe un 1 desde el exterior de la UC.

La unidad de control de cualquier máquina de Turing también tiene cinco *retardos* para ayudar a retrasar, por un tiempo constante, el paso de ciertas señales a los componentes con los que estén conectados. Estos están representados por pequeños triángulos etiquetados del 1 al 5 (ver figura 7.5). Su numeración sirve para indicar el orden en el que deben de liberarse para lograr un correcto funcionamiento en toda la MT, entonces el retardo-1 será primero en liberarse y el retardo-5 será el último.

Técnicamente un retardo está diseñado por un canal doble (ver figura 6.19) de longitud constante que es proporcional al tiempo de liberación de la señal. Cada retardo tiene asignado un tiempo distinto, por lo tanto, la longitud del canal que representa a cada uno de ellos también es distinta.

A continuación se describe la funcionalidad de cada retardo:

- Retardo-1: Antes de la activación de este retardo, cada $\texttt{q}_\texttt{i}$ ya tiene su valor correspondiente asignado y además en el *flip-flop* `D` ya se almacenó el dato enviado desde $\texttt{DB}_0$, mismo que activa el cálculo de las compuertas que representan a las fórmulas lógicas. Entonces, el retardo-1 deja pasar a la señal $\texttt{DP}_0$ después de un tiempo t_1. Este tiempo asegura que las señales resultantes de las fórmulas lógicas hayan sido debidamente calculadas y colocadas en las entradas de los *flip-flops* del conjunto estatus nuevo.

- Retardo-2: Deja pasar a la señal $\texttt{DP}_0$ después de un tiempo t_2, tal que $t_2 > t_1$. Este tiempo asegura que la señal de cada *flip-flop* del conjunto estatus nuevo haya sido debidamente calculada y transferida a las entradas de los *flip-flops* del conjunto estado actual.

- Retardo-3: Deja pasar a la señal $\texttt{DP}_0$ después de un tiempo t_3, tal que $t_3 > t_2$. Este tiempo asegura que la actualización de los valores en el conjunto estatus actual ya se haya llevado a cabo.

- Retardo-4: Deja pasar la señal proveniente de la compuerta `AND` etiquetada por $\texttt{A}_4$ (ver figura 7.5) después de un tiempo t_4. Este tiempo permite que las señales

W_0 y DF_0 se adelanten ligeramente hacia la i-ésima celda (en la que se encuentra presente la cabeza) antes de enviar a las señales MB_0 y MF_0 hacia la misma.

- Retardo-5: Deja pasar a la señal proveniente de A_4 después de un tiempo t_5, tal que $t_5 > t_4$. Este tiempo asegura que el movimiento de cabeza ya se haya realizado en la i-ésima celda (que contiene la cabeza) antes de enviar nuevamente la señal de lectura.

El conjunto de compuertas AND y OR, etiquetadas respectivamente por A_i y O_i, en la unidad de control (ver figura 7.5) terminan de concretar las señales de salida que se envían a la celda cero. Su etiquetación ayuda a describir el funcionamiento de la UC en cada proceso de la máquina de Turing, como se verá más adelante.

7.3. Celda cero

La activación de la celda cero es otra forma de detener la ejecución de la máquina de Turing, ya que indica que la cabeza simulada se ha caído a la izquierda de la cinta (recuérdese que la cinta es infinita sólo hacia la derecha). Esto podría suceder si por ejemplo, la cadena de entrada es $w = 001$ y la MT es la descrita en la tabla 7.3. Entonces, las descripciones instantáneas involucradas son: $q_0001 \vdash 0q_101 \vdash q_1011 \vdash 111$. Nótese que la última descripción instantánea es excepcional ya que no contiene ningún estado, lo cual indica que la cabeza se cayó por la izquierda de la cinta.

El diseño interno de la celda cero es sencillo (ver figura 7.6) ya que consta de:

1. Un *flip-flop* SR, etiquetado por H (Head Here), para almacenar la señal indicadora de que la cabeza de la máquina de Turing está en esta celda.

2. Un conjunto de compuertas lógicas AND y NOT, etiquetadas respectivamente por A_i y N_i para explicar el funcionamiento de la celda cero en la ejecución de la máquina de Turing. Cabe mencionar que el componente etiquetado por N_1 es una compuerta NOT y no un retardo como en la unidad de control.

3. Un conjunto de señales de entrada (R_0, W_0, DF_0, MB_0, MF_0, HB_1, DB_1 y DP_1) y de salida (R_1, W_1, DF_1, MB_1, MF_1, HF_1, DB_0 y DP_0) para comunicarse con la unidad de control y con la siguiente celda de memoria (1-ésima celda). Las señales conectadas a la unidad de control tienen subíndice 0 y las que están conectadas a la 1-ésima celda tienen subíndice 1.

De manera general, HF (Head Forward) y HB (Head Backward) son señales locales de las celdas de memoria que permiten ejecutar el movimiento de la cabeza indicado por MF y MB, respectivamente.

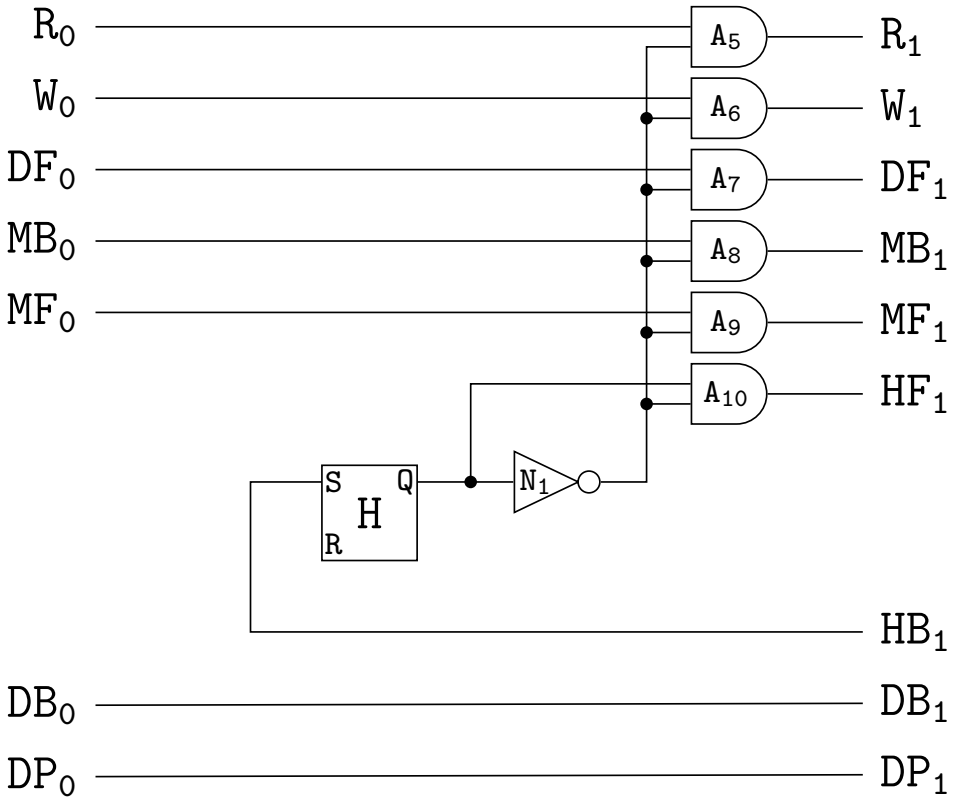


Figura 7.6: Diseño de la celda cero de memoria para cualquier máquina de Turing.

H puede almacenar un 1 o un 0 para indicar respectivamente si la cabeza está o no en la celda cero. Nótese que si sucede el primer caso, la señal de N_1 provocará que las compuertas AND envíen ceros hacia las señales de salida que están conectadas a la siguiente (1-ésima) celda de memoria, lo cual impedirá que cualquier proceso en la máquina de Turing continue, es decir, provocará la detención de la ejecución de ésta. En caso contrario, N_1 dejará que las señales que llegaron de la unidad de control, pasen a la siguiente celda de memoria.

La compuerta A_{10} siempre alimenta con ceros a la señal HF_1 pues la cabeza de lectura nunca pasará de la celda cero a la celda uno (ni a ninguna otra celda).

7.4. i-ésima celda

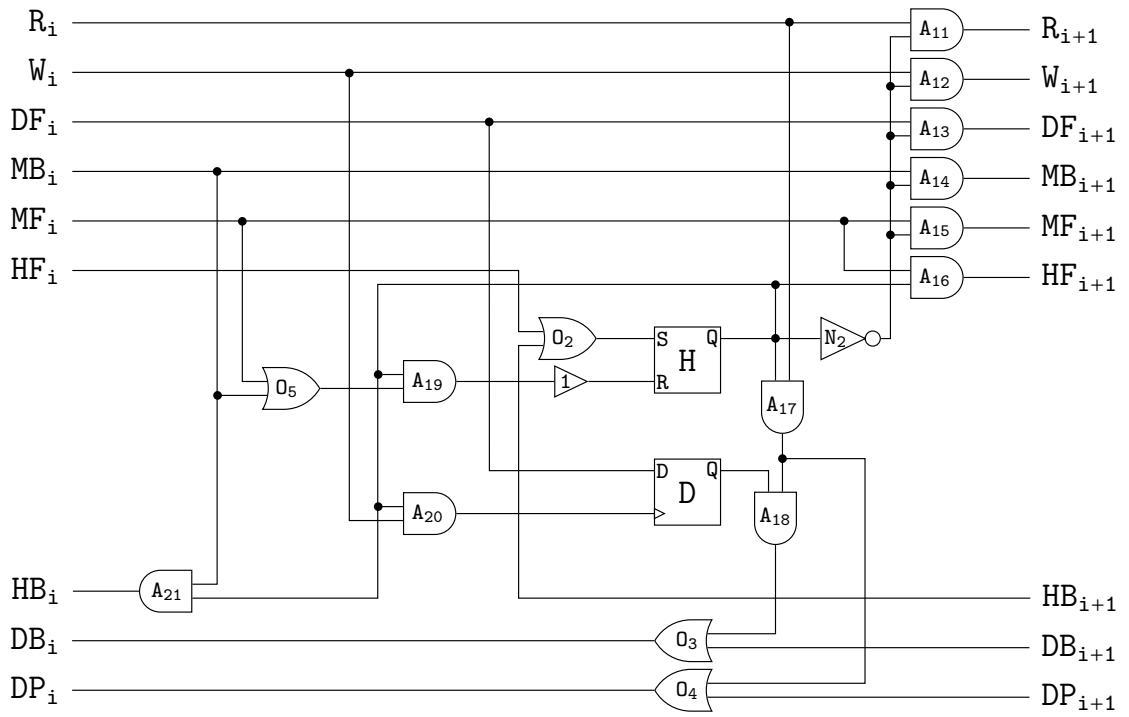


Figura 7.7: Diseño de la i-ésima celda de memoria ($i > 0$) para cualquier máquina de Turing.

Las i-ésimas celdas (ver figura 7.7) son aquellas que conforman a la cinta infinita de la máquina de Turing (no se contempla a la celda cero). Cada una de la celdas consta de:

1. Un *flip-flop* SR, etiquetado por D (Data), para almacenar el dato.

2. Un *flip-flop* tipo D, etiquetado por H (Head Here), para guardar la señal que indica si la cabeza está o no apuntando a esta celda (1 cuando sí y 0 cuando no).

3. Un conjunto de compuertas AND, OR y NOT etiquetadas respectivamente por A_i, O_i y N_i para atender la señales recibidas en la celda y controlar el funcionamiento de la misma.

4. Un conjunto de señales de entrada (R_i, W_i, DF_i, MB_i, MF_i, HF_i, HB_{i+1}, DB_{i+1} y DP_{i+1}) y de salida (R_{i+1}, W_{i+1}, DF_{i+1}, MB_{i+1}, MF_{i+1}, HF_{i+1}, HB_i, DB_i y DP_i) para comunicarse con la previa y posterior celda de memoria. Las señales conectadas a la celda $i-1$ tienen subíndice i y las que están conectadas a la celda $i+1$ tienen subíndice $i+1$.

Las compuertas en la i-ésima celda tienen una finalidad específica, a saber:

- N_2: Si la cabeza está en la celda i, entonces envía la señal que bloquea a las señales R_{i+1}, W_{i+1}, DF_{i+1}, MB_{i+1}, MF_{i+1} y HF_{i+1}; de lo contrario, permite el paso de las señales que llegaron de la celda $i-1$ hasta llegar a la celda donde está la cabeza.

- A_{17}: Si la señal de lectura está activa y la cabeza está en la i-ésima celda, entonces induce a leer el dato almacenado en D, así como a encender la señal DP_i.

- O_3: Envía el dato almacenado en D a través de la señal DB_i. O bien, deja pasar la señal del dato proveniente de DB_{i+1} que viene de alguna celda posterior.

- O_4: Envía la señal de dato presente (DP_i) que indica que la señal DB_i contiene un dato válido en ese momento. O bien, deja pasar la señal de dato presente DP_{i+1} que viene de alguna celda posterior.

- A_{20}: Genera la señal que permite almacenar el dato (enviado por la unidad de control) en la i-ésima celda de memoria, específicamente en el *flip-flop* D.

- A_{21}: Envía la señal que indica que la cabeza debe ser activada en la celda anterior.

- A_{16}: Envía la señal que indica que la cabeza debe ser activada en la celda siguiente.

- O_5: Envía la primera señal para inducir que la cabeza en la i-ésima celda se apague.

- A_{19}: Envía la señal para apagar la cabeza en la i-ésima celda.

- Retardo-1: Libera la señal recibida de $\mathtt{A}_{19}$ para que pasado un cierto tiempo t la cabeza se apague en la i-ésima celda. Esto para asegurar que la señal de la cabeza activa se use de manera adecuada en las demás compuertas.

- $\mathtt{O}_2$: Permite el paso de la señal que indica que la cabeza tiene que activarse en la i-ésima celda.

7.5. Simulación de la ejecución de la máquina de Turing

La simulación de la ejecución de la máquina de Turing se lleva a cabo mediante la sucesión de cuatro procesos: lectura, cambio de estado, escritura y movimiento de cabeza. Estos se repiten indefinidamente de manera cíclica (ver figura 7.8) a menos que la máquina de Turing llegue a una forma de paro.

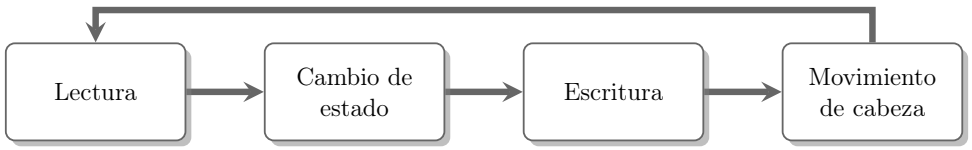


Figura 7.8: Sucesión cíclica de los procesos que se realizan en la simulación de la ejecución de la máquina de Turing.

En las siguientes secciones se describe la ejecución de cada proceso y se indica la configuración inicial que debe de tener la unidad de control y las celda de memoria en cada uno de ellos para una correcta funcionalidad. Se recomienda ver en todo momento las figuras 7.5, 7.6 y 7.7 para identificar las etiquetas usadas en la explicación.

7.5.1. Lectura

Es el proceso que se encarga de tomar el dato almacenado en la celda de memoria, donde se encuentra localizada la cabeza, para después enviarlo a la unidad de control.

El proceso de lectura requiere que la unidad de control y las celdas de memoria tengan cierta configuración inicial. Para la unidad de control se consideran dos posibles configuraciones dependiendo el caso en el que se esté: 1) Cuando ejecución de la máquina de Turing está por comenzar (ver tabla 7.4) y 2) cuando la ejecución ya había comenzado previamente (ver tabla 7.5).

Unidad de control					
	Conjunto estatus actual	`Start`	`Hlt`	Señales de entrada	Señales de salida
Valor inicial	Todos los *flip-flops* almacenan un 0 a excepción del correspondiente al estado incial de la MT.	1	0	0	0

Tabla 7.4: Configuración inicial de la unidad de control en el proceso de lectura para cuando la ejecución de la máquina de Turing está por comenzar. El valor de los *flip-flops* del conjunto estado nuevo no es importante.

Unidad de control						
	Conjunto estatus actual	`Start`	`Hlt`	Retardo-5	Señales de entrada	Señales de salida
Valor inicial	El valor del *flip-flop* `D` no es importante. El resto de los *flip-flops* almacenan un 0 a excepción del correspondiente al estado actual en que se encuentra la MT.	0	0	1	0	0

Tabla 7.5: Configuración inicial de la unidad de control en el proceso de lectura para cuando la ejecución de la máquina de Turing ya había comenzado, es decir, cuando el proceso anterior fue el movimiento de cabeza. El valor de los *flip-flops* del conjunto estado nuevo no es importante.

En la tabla 7.6 se muestra la configuración inicial de la celda cero y de cualquier i-ésima celda, según sea el estatus actual de la máquina de Turing. Si la cabeza está en dicha celda entonces `H` debe almacenar un 1; de lo contrario un 0. En esta tabla se usa el símbolo * para representar de manera general cualquiera de estos dos casos. Mientras que d indica el valor del dato almacenado en el *flip-flop* `D`, el cual puede ser 0 o 1. En el caso de la celda cero, el dato d no existe.

Celdas de memoria							
	Celda cero			i-ésima celda			
	H	Señales de entrada	Señales de salida	D	H	Señales de entrada	Señales de salida
Valor inicial	*	0	0	d	*	0	0

Tabla 7.6: Configuración inicial de la celda cero y de cualquier i-ésima celda en el proceso de lectura. El asterisco (*) en H hace referencia que puede estar un 1 o un 0 según si la cabeza está o no en la celda, respectivamente. Mientras que d (dato almacenado en la celda) puede ser 0 o 1.

La lectura comienza en la unidad de control con la activación de la señal $\mathtt{R_0}$. Esta puede activarse mediante:

1. La señal `Start`, siempre y cuando la ejecución de la máquina de Turing esté por comenzar, o por
2. la señal liberada del retardo-5, siempre y cuando el proceso anterior haya sido el movimiento de cabeza. Esta señal proviene de la compuerta $\mathtt{A_4}$ como resultado de procesar la información saliente de `Hlt` junto con la liberada del retardo-3.

Posteriormente, se envía la señal de lectura ($\mathtt{R_0}$) desde la unidad de control hacia la cinta, donde se busca la señal que indica que la cabeza está activa en alguna celda de memoria. De aquí surgen cuatro casos:

1. Si la cabeza no está en la celda cero, la compuerta $\mathtt{N_1}$ envía unos a las compuertas $\mathtt{A_5}, \mathtt{A_6}, \mathtt{A_7}, \mathtt{A_8}$, $\mathtt{A_9}$ y $\mathtt{A_{10}}$ para que éstas dejen pasar las señales que recibieron en sus otras entradas. En particular, la señal $\mathtt{R_0}$ pasa a la siguiente celda con el valor que trae desde la unidad de control. Por el propio diseño de la celda cero siempre se transfiere un 0 a la señal $\mathtt{HF_1}$ mediante la compuerta $\mathtt{A_{10}}$, lo cual implica que la cabeza nunca saldrá de esta celda una vez que haya llegado ahí.
2. Si la cabeza se encuentra en la celda cero, la compuerta $\mathtt{N_1}$ envía ceros a las compuertas $\mathtt{A_5}, \mathtt{A_6}, \mathtt{A_7}, \mathtt{A_8}$, $\mathtt{A_9}$ y $\mathtt{A_{10}}$ para que éstas bloqueen el paso de las señales que recibieron en sus otras entradas. En este caso, la cabeza ya nunca saldrá de esta celda y el proceso se detiene.
3. Si la cabeza no está en la celda i, la compuerta $\mathtt{N_2}$ envía unos a las compuertas $\mathtt{A_{11}}, \mathtt{A_{12}}, \mathtt{A_{13}}, \mathtt{A_{14}}$ y $\mathtt{A_{15}}$ para que éstas dejen pasar las señales que recibieron en sus otras entradas. En particular, la señal $\mathtt{R_i}$ pasa a la siguiente celda con el valor que trae desde la unidad de control. Mientras que las compuertas $\mathtt{A_{16}}$ y $\mathtt{A_{21}}$ generan un 0 que es transferido respectivamente hacia $\mathtt{HF_{i+1}}$ y $\mathtt{HB_i}$.

4. Si la cabeza es hallada en la i-ésima celda, se desactiva la señal $\texttt{R}_{\texttt{i+1}}$ y después se envía a la unidad de control el dato almacenado en $\texttt{D}$ (a través de $\texttt{DB}_{\texttt{i}}$) junto con la señal $\texttt{DP}_{\texttt{i}}$.

   En este punto se debe de tener en cuenta que el valor del *flip-flop* $\texttt{H}$ no ha cambiado en largo tiempo (desde una lectura anterior o nunca, si es que no se ha comenzado la ejecución de la máquina de Turing) y que por lo tanto, su valor ya se ha propagado hacia las compuertas $\texttt{N}_{\texttt{2}}$, $\texttt{A}_{\texttt{16}}$ y $\texttt{A}_{\texttt{21}}$. Entonces, también se considera que la señal de $\texttt{N}_{\texttt{2}}$ ya se ha propagado hasta las compuertas $\texttt{A}_{\texttt{11}}, \texttt{A}_{\texttt{12}}, \texttt{A}_{\texttt{13}}, \texttt{A}_{\texttt{14}}$ y $\texttt{A}_{\texttt{15}}$ para que éstas bloqueen el paso de las señales que están recibiendo en sus otras entradas, en particular, a la señal $\texttt{R}_{\texttt{i}}$ en el momento en que llega a esta celda.

   Entonces en el momento en que llega la señal $\texttt{R}_{\texttt{i}}$ a la i-ésima celda, se realizan las siguientes operaciones:

   a) La compuerta $\texttt{A}_{\texttt{17}}$ procesa la información recibida de $\texttt{H}$ y de $\texttt{R}_{\texttt{i}}$.

   b) La compuerta $\texttt{A}_{\texttt{11}}$ genera un cero para bloquear la propagación de la señal $\texttt{R}_{\texttt{i}}$ a la siguiente celda.

   c) La compuerta $\texttt{A}_{\texttt{18}}$ procesa la información recibida de $\texttt{D}$ y de $\texttt{A}_{\texttt{17}}$.

   d) La compuerta $\texttt{O}_{\texttt{3}}$ permite el paso de la señal recibida de $\texttt{A}_{\texttt{18}}$.

   e) La compuerta $\texttt{O}_{\texttt{4}}$ permite el paso de la señal recibida de $\texttt{A}_{\texttt{17}}$.

   Las señales $\texttt{DB}_{\texttt{i}}$ y $\texttt{DP}_{\texttt{i}}$ se envían hasta la unidad de control. Esto implica que ambas deben de pasar por todas las celdas de memoria que se encuentran a la izquierda de la celda i, incluyendo a la celda cero, para llegar al destino correspondiente. En la celda cero, $\texttt{DB}_{\texttt{i}}$ y $\texttt{DP}_{\texttt{i}}$ pasan de manera directa hacia $\texttt{DB}_{\texttt{0}}$ y $\texttt{DP}_{\texttt{0}}$, respectivamente. Mientras que en las demás celdas, las compuertas $\texttt{O}_{\texttt{3}}$ y $\texttt{O}_{\texttt{4}}$ dejan pasar a $\texttt{DB}_{\texttt{i}}$ y $\texttt{DP}_{\texttt{i}}$, respectivamente, hacia $\texttt{DB}_{\texttt{i-1}}$ y $\texttt{DP}_{\texttt{i-1}}$.

   En el momento en que señales $\texttt{DB}_{\texttt{i}}$ y $\texttt{DP}_{\texttt{i}}$ llegan a la unidad de control se termina el proceso de lectura.

7.5.2. Cambio de estado

Este proceso se ejecuta después de la lectura y se lleva a cabo sólo en la unidad de control. Además de la transición de estado propiamente, calcula también el dato (d') a escribir en el proceso de escritura y el tipo de movimiento de cabeza a realizar (ya sea derecha o a la izquierda) en el proceso respectivo. Posteriormente preparara a las señales $\texttt{W}_{\texttt{0}}$ y $\texttt{DF}_{\texttt{0}}$ para la escritura, a las señales $\texttt{MB}_{\texttt{0}}$ y $\texttt{MF}_{\texttt{0}}$ para el movimiento de cabeza y a la señal $\texttt{R}_{\texttt{0}}$ para el generar el proceso de lectura (sólo en caso de que la máquina de Turing aún no pare).

Esta fase requiere que la unidad de control tenga una configuración inicial (ver tabla 7.7), donde destaca que la señal $\texttt{DB}_0$ tendrá el valor d (el dato leído en la cinta previamente) y la señal $\texttt{DP}_0$ tendrá un 1 (para indicar el dato presente en la UC).

Unidad de control						
	Conjunto estatus actual	`Start`	`Hlt`	Señales de entrada		Señales de salida
				$\texttt{DB}_0$	$\texttt{DP}_0$	
Valor inicial	El *flip-flop* `D` almacena el dato d del ciclo anterior (o cero si este es el primer ciclo). Mientras que los demás *flip-flops* almacenan un 0 a excepción del correspondiente al estado actual en que se encuentra la MT.	0	0	d	1	0

Tabla 7.7: Configuración inicial de la unidad de control en el procesamiento de información. El valor del dato leído en la cinta es representado por d, el cual puede ser 0 o 1.

Este proceso comienza en el momento que las señales $\texttt{DB}_0$ y $\texttt{DP}_0$, enviadas desde la i-ésima celda, llegan a la unidad de control. Ya aquí se ejecutan las siguientes operaciones:

1. El dato (d), transportado por la señal $\texttt{DB}_0$, se posiciona en la entrada del *flip-flop* `D`.
2. El reloj del *flip-flop* `D` es activado mediante la señal $\texttt{DP}_0$ para almacenar a d.
3. El nuevo dato d (almacenado en el *flip-flop* `D`) se propaga hacia las compuertas (que representan a las fórmulas lógicas) causando que algunas de ellas cambien de estado. Estos cambios continúan su propagación hasta los *flip-flops* del conjunto estatus nuevo.
4. El retardo-1 se libera permitiendo que la señal $\texttt{DP}_0$ llegue a todos los relojes de los *flip-flops* del conjunto estatus nuevo, para después almacenar en ellos los datos posicionados previamente en sus entradas. Además se deja pasar la señal $\texttt{DP}_0$ hacia el retardo-3.
5. Los datos almacenados en los *flip-flops* $\texttt{q}'_\texttt{i}$ se transfieren a las entradas de sus respectivos $\texttt{q}_\texttt{i}$ pertenecientes al conjunto estatus actual.

6. El retardo-2 es liberado permitiendo que la señal $\texttt{DP}_0$ llegue a todos los relojes de los *flip-flops* del conjunto estatus actual, para después almacenar en ellos los datos posicionados previamente en sus entradas. Además, las salidas de estos *flip-flops* se propagan hacia las compuertas (que representan a las fórmulas lógicas) y hacia los *flip-flops* del conjunto estado nuevo. Estos nuevos datos serán usados hasta el siguiente ciclo en la ejecución de la máquina de Turing.

7. El retardo-3 es liberado permitiendo pasar la señal $\texttt{DP}_0$ hacia la compuerta $\texttt{A}_4$.

8. La señal generada en la compuerta $\texttt{A}_4$ es transferida hacia el retardo-4 y retardo-5, así como a la señal $\texttt{W}_0$ y a una de las entradas de la compuerta $\texttt{A}_1$.

9. La compuerta $\texttt{A}_1$ procesa la señal recibida previamente junto con el dato almacenado en $\texttt{D}'$ y envía el resultado hacia la señal $\texttt{DF}_0$.

10. El retardo-4 es liberado permitiendo pasar la señal de $\texttt{A}_4$ hacia las compuertas $\texttt{A}_2$ y $\texttt{A}_3$.

11. La compuerta $\texttt{A}_2$ procesa la señal recibida previamente junto con la negación del dato almacenado en $\texttt{MF}$ y envía el resultado hacia la señal $\texttt{MB}_0$.

12. La compuerta $\texttt{A}_3$ procesa la señal recibida previamente junto con del dato almacenado en $\texttt{MF}$ y envía el resultado hacia la señal $\texttt{MF}_0$.

13. El retardo-5 es liberado permitiendo pasar la señal de $\texttt{A}_4$ hacia la compuerta $\texttt{O}_1$. En caso de que $\texttt{A}_4$ genere un 1, el proceso de lectura volverá a realizarse por lo que $\texttt{R}_0$ se activará nuevamente; de lo contrario la máquina de Turing detendrá su ejecución.

Nótese que las operaciones del punto 3 permiten el cálculo del nuevo estado, del dato a escribir y del movimiento de cabeza a realizar. Además, las que están en los puntos 8 y 9 preparan a las señales para el proceso de escritura; mientras que las de los puntos 10, 11 y 12 preparan a las señales para realizar el movimiento de cabeza.

Es importante mencionar que las señales de escritura ($\texttt{W}_0$ y $\texttt{DF}_0$) salen de manera conjunta de la unidad de control y que posteriormente, después de la liberación del retardo-4, salen al mismo tiempo las señales ($\texttt{MB}_0$ y $\texttt{MF}_0$) correspondientes al movimiento de cabeza. De esta manera las señales de ambos procesos van en tándem por el bus de salida con el retraso adecuado para dar tiempo a que cada proceso termine antes de comenzar el siguiente. Si la ejecución de la máquina de Turing aún no termina, la señal de lectura también saldrá al bus de salida después de cierto tiempo de la señales del proceso del movimiento de cabeza.

7.5.3. Escritura

Es el proceso que se encarga de escribir el dato en celda donde está la cabeza de la máquina de Turing. La escritura requiere de una configuración inicial para la unidad de control (ver tabla 7.8) y para las celdas de memoria (ver tabla 7.9). Donde d' indica el dato a escribir en el *flip-flop* D, el cual puede ser 0 o 1. Mientras que el símbolo * representa si la cabeza está o no el la celda, según sea el caso.

Unidad de control									
	Conjunto estatus actual	Start	Hlt	Señales de entrada	Señales de salida				
					R_0	W_0	DF_0	MB_0	MF_0
Valor inicial	El *flip-flop* D almacena el dato d del ciclo anterior (o cero si este es el primer ciclo). Mientras que los demás *flip-flops* almacenan un 0 a excepción del correspondiente al estado al que transitó la MT.	0	0	0	0	1	d'	0	0

Tabla 7.8: Configuración inicial de la unidad de control en el proceso de escritura. El valor a escribir en la cinta es representado por d', el cual puede ser 0 o 1.

La escritura comienza desde la unidad de control con la activación de W_0. Esta señal se envía junto con DF_0 a la cinta para escribir el dato d' en la i-ésima celda donde este posicionada la cabeza. En este translado de señales se presentan tres casos:

1. Si la cabeza no está en la celda cero, la compuerta N_1 envía unos a las compuertas A_5, A_6, A_7, A_8, A_9 y A_{10} para que éstas dejen pasar las señales que recibieron en sus otras entradas. En particular, las señales W_0 y DF_0 pasan a la siguiente celda con el valor que traen desde la unidad de control. Por el propio diseño de la celda cero siempre se transfiere un 0 a la señal HF_1.

2. Si la cabeza no está en la celda i, la compuerta N_2 envía unos a las compuertas $A_{11}, A_{12}, A_{13}, A_{14}$ y A_{15} para que éstas dejen pasar las señales que recibieron en sus otras entradas. En particular, las señales W_i y DF_i pasan a la siguiente celda con el valor que traen desde la unidad de control. Por la configuración inicial de la i-ésima celda, de las compuertas A_{16} y A_{21} se obtiene un 0 que es transferido hacia HF_{i+1} y HB_i, respectivamente.

Celdas de memoria							
	Celda cero			i-ésima celda			
	H	Señales de entrada	Señales de salida	D	H	Señales de entrada	Señales de salida
Valor inicial	0	0	0	d	*	0	0

Tabla 7.9: Configuración inicial de la celda cero y de cualquier i-ésima celda en el proceso de escritura. El asterisco (*) en H hace referencia que puede estar un 1 o un 0 según si la cabeza está o no en la celda, respectivamente. Mientras que d es el valor del dato almacenado en la celda, el cual puede ser 0 o 1. Recuérdese que en caso de que H tenga valor 1 será porque la cabeza está en la celda cero, y entonces, la señal W_0 se bloqueará mediante la compuerta A_6 (ver figura 7.6) y todo el proceso acabará, es decir, la máquina de Turing se detendrá.

3. Si la cabeza está en la i-ésima celda se bloquea el paso de la señal W_{i+1} y se bloquea el paso de d' hacia DF_{i+1}. Posteriormente se almacena el dato d' en el *flip-flop* D de la celda i.

   En este punto se debe de tener en cuenta que sólo las señales W_i y DF_i han llegado desde la celda previa, y que dado que la señal de N_2 ya se ha propagado previamente hacia las compuertas $A_{11}, A_{12}, A_{13}, A_{14}$ y A_{15}, dichas señales serán bloqueadas hacia la $i+1$ celda. Además, se considera que la señal R_i ya no está activa porque sólo es un pulso al momento que se envía la señal de lectura.

   Entonces, las operaciones que se ejecutan a la llegada de W_i y DF_i son las siguientes:

   a) La compuerta A_{13} envía un 0 hacia la señal DF_{i+1}.

   b) La compuerta A_{12} envía un 0 hacia la señal W_{i+1}.

   c) El dato d' es colocado en la entrada del fli-flop D.

   d) La compuerta A_{20} procesa la información reciba de H y de W_i.

   e) La señal resultante de la compuerta A_{20} se envía al reloj del fli-flop D para almacenar d' en este mismo.

   En el momento en que el dato d' es almacenado en el *flip-flop* D termina el proceso de escritura.

7.5.4. Movimiento de cabeza

Este proceso se realiza poco tiempo después de que la escritura haya terminado. En él se considera que la cabeza de la máquina de Turing se encuentra en una i-ésima

celda de memoria para moverla hacia la izquierda (a la $\{i-1\}$-ésima celda) o hacia la derecha (a la $\{i+1\}$-ésima celda).

Antes de iniciar el movimiento de cabeza se requiere de una configuración inicial para la unidad de control (ver tabla 7.10) y para las celdas de memoria. En este caso, la celda cero poseé una configuración idéntica a como se muestra en el proceso de escritura (ver tabla 7.9). Las celdas que no sean la i-ésima celda (donde está la cabeza) deben de tener una configuración diferente a la de la celda i (ver tabla 7.11). Esto se debe a que las señales de este proceso se envían un poco tiempo después de las señales de escritura, entonces cuando se llegue a la celda i, ésta estará recientemente modificada.

Unidad de control									
	Conjunto estatus actual	Start	Hlt	Señales de entrada	Señales de salida				
					R_0	W_0	DF_0	MB_0	MF_0
Valor inicial	Todos los *flip-flops* almacenan un 0 a excepción del correspondiente al estado al que transitó la MT	0	0	0	0	0	0	m	$\neg m$

Tabla 7.10: Configuración inicial de la unidad de control en el proceso movimiento de cabeza. Donde $\neg m$ es la negación de m y además m puede tomar el valor 0 o 1, según sea el movimiento que se va a realizar.

Celdas de memoria								
	Celdas distintas a la $\{i\}$-ésima celda				i-ésima celda			
	D	H	Señales de entrada	Señales de salida	D	H	Señales de entrada	Señales de salida
Valor inicial	$*$	0	0	0	d'	1	0	0

Tabla 7.11: Configuración inicial en el proceso movimiento de cabeza de la i-ésima celda (lugar donde se localiza la cabeza de la máquina de Turing) y del resto de las celdas. El dato que se escribió recientemente en la i-ésima celda se denota por d', mientras que el símbolo $*$ representa a el valor del dato almacenado en el resto de las celdas. Tanto $*$ como d' pueden tomar valor 0 o 1, según sea el caso.

El movimiento de cabeza comienza desde la unidad de control con la activación de la señal MB_0 (si la cabeza se moverá hacia la izquierda) o MF_0 (si la cabeza se

moverá hacia la derecha). Ambas señales se envían al mismo tiempo hacia la i-ésima celda donde está posicionada la cabeza para realizar el movimiento respectivo. En la propagación de estas señales se presentan tres casos:

1. Al paso por la celda cero, la compuerta $\mathtt{N_1}$ envía unos a las compuertas $\mathtt{A_5}, \mathtt{A_6}$, $\mathtt{A_7}, \mathtt{A_8}$, $\mathtt{A_9}$ y $\mathtt{A_{10}}$ para que éstas dejen pasar las señales que recibieron en sus otras entradas. En particular, las señales $\mathtt{MB_0}$ y $\mathtt{MF_0}$ pasan a la siguiente celda con el valor que traen desde la unidad de control. Por el propio diseño de la celda cero siempre se transfiere un 0 a la señal $\mathtt{HF_1}$.

2. Si la cabeza no está en la celda i, la compuerta $\mathtt{N_2}$ envía unos a las compuertas $\mathtt{A_{11}}, \mathtt{A_{12}}, \mathtt{A_{13}}, \mathtt{A_{14}}$ y $\mathtt{A_{15}}$ para que éstas dejen pasar las señales que recibieron en sus otras entradas. En particular, las señales $\mathtt{MB_i}$ y $\mathtt{MF_i}$ pasan a la siguiente celda con el valor que traen desde la unidad de control. Por otra parte, las compuertas $\mathtt{A_{16}}$ y $\mathtt{A_{21}}$ generan un 0 que es transferido respectivamente hacia $\mathtt{HF_{i+1}}$ y $\mathtt{HB_i}$.

3. Si la cabeza está en la i-ésima celda se bloquea el paso de las señales correspondientes hacia $\mathtt{MB_{i+1}}$ y $\mathtt{MF_{i+1}}$. Además, se activa la señal que indica si se encenderá la cabeza de la celda izquierda ($\mathtt{HB_i}$) o de la celda derecha ($\mathtt{HF_{i+1}}$), para después apagar la señal de la cabeza en la celda i. Posteriormente, dependiendo del movimiento que se debe realizar, se activa la cabeza en la celda $i-1$ o en la celda $i+1$.

   En este punto se debe de tener en cuenta que sólo las señales $\mathtt{MB_i}$ y $\mathtt{MF_i}$ han llegado desde la celda previa, y que dado que la señal de $\mathtt{N_2}$ ya se ha propagado previamente hacia las compuertas $\mathtt{A_{11}}, \mathtt{A_{12}}, \mathtt{A_{13}}, \mathtt{A_{14}}$ y $\mathtt{A_{15}}$, dichas señales serán bloqueadas hacia la $i+1$ celda. También se considera que la señal del *flip-flop* $\mathtt{H}$ ya ha llegado a las compuertas $\mathtt{A_{16}}$ y $\mathtt{A_{21}}$. Por otra parte, no hay que olvidar que las señales $\mathtt{W_i}$ y $\mathtt{DF_i}$ ya no están activas porque sólo son pulsos al momento en que envía la señal de escritura.

   Entonces, las operaciones que se ejecutan a la llegada de $\mathtt{MB_i}$ y $\mathtt{MF_i}$ son las siguientes:

   a) La compuerta $\mathtt{A_{16}}$ procesa la información recibida de $\mathtt{MF_i}$. Si la señal $\mathtt{MF_i}$ está activa, entonces, la compuerta $\mathtt{A_{16}}$ envía un 1 hacia la señal $\mathtt{HF_{i+1}}$; de lo contrario envía un 0. La señal $\mathtt{HF_{i+1}}$ indica a la celda siguiente que su cabeza tiene que ser activada.

   b) La compuerta $\mathtt{A_{21}}$ procesa la información recibida de $\mathtt{MB_i}$. Si la señal $\mathtt{MB_i}$ está activa, entonces, la compuerta $\mathtt{A_{21}}$ envía un 1 hacia la señal $\mathtt{HB_i}$; de lo contrario envía un 0. La señal $\mathtt{HB_i}$ indica a la celda previa que su cabeza tiene que ser activada.

c) La compuerta $\mathtt{O_5}$ procesa la información recibida de $\mathtt{MB_i}$ y de $\mathtt{MF_i}$. Aquí se genera un 1 que es enviado a $\mathtt{A_{19}}$.

d) La compuerta $\mathtt{A_{19}}$ procesa la información recibida del inciso anterior junto con la señal de H y envía un 1 al retardo-1 de esta i-ésima celda.

e) El retardo-1 libera la señal contenida y lo envía a la entrada de reset (R) en el *flip-flop* H para apagar la señal de la cabeza en la celda i. Esto sucede una vez que la señal H haya sido usada para todos los propósitos previamente mencionados.

f) En caso de que la señal $\mathtt{HB_i}$ se haya activado, entonces en la celda previa $(i-1)$, la compuerta $\mathtt{O_2}$ procesa las señales $\mathtt{HF_{i-1}}$ (que está inactiva) y $\mathtt{HB_i}$ (que está activa) y envía el resultado a la entrada set (S) del *flip-flop* H para activar la cabeza de la celda $i-1$.

g) En caso de que la señal $\mathtt{HF_{i+1}}$ se haya activado, entonces en la celda siguiente $(i+1)$, la compuerta $\mathtt{O_2}$ procesa las señales $\mathtt{HF_{i+1}}$ (que está activa) y $\mathtt{HB_{i+2}}$ (que está inactiva) y envía el resultado a la entrada set (S) del *flip-flop* H para activar la cabeza de la celda $i+1$.

El proceso de movimiento de cabeza finaliza cuando se ejecutan las instrucciones del inciso f o g, dependiendo del movimiento que se haya realizado.

7.6. Ejemplo de simulación de la ejecución de la máquina de Turing

Se tomará el ejemplo de la máquina de Turing definida en la tabla 7.3 para mostrar las instrucciones que se realizan en cada proceso de la simulación tanto en la unidad de control como en las celdas de memoria. Para esto se usarán tablas con cuatro secciones:

- Señales de entrada: contiene todas las señales de entrada (junto con sus respectivos valores) que tiene la unidad de control o las celdas de memoria, según sea el caso.
- Valores internos: contiene sólo los *flip-flops* (con su respectivo valor almacenado) que son relevantes en el estatus previo a la modificación del proceso correspondiente.
- Cambios internos: contiene sólo los componentes (compuertas y *flip-flops*) que son modificados durante el proceso correspondiente. Se considera un cierto orden de modificación, por lo que el componente etiquetado con un orden 1 será el primero en ejecutarse, y después en cascada, el 2 será el segundo, etc.

- Señales de salida: contiene todas las señales de salida (junto con sus respectivos valores) que tiene la unidad de control o las celdas de memoria, según sea el caso.

Para el ejemplo de la simulación se considerará que la ejecución de la máquina de Turing está por iniciar y que en la cinta está almacenada la cadena w = 10111, entonces la cabeza está apuntando al primer 1 (de izquierda a derecha) de w, como se puede ver en la figura 7.1.

Recuérdese ver en todo momento las figuras 7.5, 7.6 y 7.7 para identificar las etiquetas usadas en las tablas.

7.6.1. Lectura

Dado que la ejecución de la máquina de Turing está por comenzar, en la unidad de control la señal `Start` es estimulada con un 1 (es decir, con un pulso) para iniciar con el proceso de lectura. En este caso, el *flip-flop* q_0 almacena un 1 debido a que es el estado inicial de la MT en cuestión (ver tabla 7.12). Este valor se matendrá en q_0 hasta que se modifique en el proceso de cambio de estado.

Unidad de control			
	Comienzo de ejecución de la MT		
	Orden	Señal/ Componente	Valor
Entradas		Start	1
		DB_0	0
		DP_0	0
Valores internos		q_0	1
		q_1	0
		q_2	0
		Hlt	0
Cambios internos	1	O_1	1
Salidas		R_0	1
		W_0	0
		DF_0	0
		MB_0	0
		MF_0	0

Tabla 7.12: Señales y componentes involucrados en la unidad de control para el proceso de lectura del primer símbolo de la cadena w.

El *flip-flop* H de la celda cero debe almacenar un 0 ya que la cabeza está inicialmente apuntando a la celda 1 de la cinta. Por lo tanto, sólo se activa la señal de lectura para enviarla a la 1-ésima celda (ver tabla 7.13).

Celda cero			
	Orden	Señal/ Componente	Valor
Entradas		R_0	1
		W_0	0
		DF_0	0
		MB_0	0
		MF_0	0
		HB_1	0
		DB_1	0
		DP_1	0
Valores internos		H	0
Cambios internos	1	A_5	1
Salidas		R_1	1
		W_1	0
		DF_1	0
		MB_1	0
		MF_1	0
		HF_1	0
		DB_0	0
		DP_0	0

Tabla 7.13: Señales y componentes involucrados en la celda cero para cuando se está yendo hacia la celda 1 para leer el primer símbolo de w.

En la celda $i = 1$ se bloquea la señal de lectura hacia la siguiente celda y se procede a leer el dato d almacenado en el *flip-flop* D. En este caso, $d = 1$ porque $w = 10111$. La señal DB_i se envía a d y DP_i se activa para indicar que el dato ha sido leído (ver tabla 7.14). Estas dos señales son enviadas hacia la unidad de control.

Cuando las señales DB_i y DP_i pasan a la celda cero son transferidas directamente hacia las señales DB_0 y DP_0, respectivamente, sin que otro componente se vea involucrado (ver tabla 7.15). Por esto, la fila correspondiente a cambios internos en la tabla 7.15 está vacía.

i-ésima celda			
	Orden	Señal/ Componente	Valor
Entradas		R_i	1
		W_i	0
		DF_i	0
		MB_i	0
		MF_i	0
		HF_i	0
		HB_{i+1}	0
		DB_{i+1}	0
		DP_{i+1}	0
Valores internos		H	1
		D	1
Cambios internos	1	A_{17}	1
	2	A_{11}	0
	3	O_3	1
	4	O_4	1
Salidas		R_{i+1}	0
		W_{i+1}	0
		DF_{i+1}	0
		MB_{i+1}	0
		MF_{i+1}	0
		HF_{i+1}	0
		HB_i	0
		DB_i	1
		DP_i	1

Tabla 7.14: Señales y compuertas involucradas en la i-ésima celda (i = 1) para el proceso de lectura del primer símbolo de w. En este caso el dato almacenado en el *flip-flop* D tiene valor 1.

Celda cero			
	Orden	Señal/ Componente	Valor
Entradas		$\mathtt{R}_0$	0
		$\mathtt{W}_0$	0
		$\mathtt{DF}_0$	0
		$\mathtt{MB}_0$	0
		$\mathtt{MF}_0$	0
		$\mathtt{HB}_1$	0
		$\mathtt{DB}_1$	1
		$\mathtt{DP}_1$	1
Valores internos		$\mathtt{H}$	0
Cambios internos			
Salidas		$\mathtt{R}_1$	0
		$\mathtt{W}_1$	0
		$\mathtt{DF}_1$	0
		$\mathtt{MB}_1$	0
		$\mathtt{MF}_1$	0
		$\mathtt{HF}_1$	0
		$\mathtt{DB}_0$	1
		$\mathtt{DP}_0$	1

Tabla 7.15: Señales y componentes involucrados en la celda cero para cuando se está enviando el dato leído (d) por la señal $\mathtt{DB}_0$ hacia la unidad de control. En este caso $d = 1$ porque la cadena de entrada es $w = 10111$.

7.6.2. Cambio de estado

Ya que se leyó un 1 en la cinta y que el estado actual es $\mathtt{q}_0$, entonces se debe de transitar al estado $\mathtt{q}_0$, escribir un 1 en la i-ésima celda y mover la cabeza a la derecha, tal como lo indica la función de transición a través de las fórmulas lógicas. Por esta razón, en los *flip-flops* $\mathtt{q}_0$, $\mathtt{D}'$ y $\mathtt{MF}$ del conjunto estatus nuevo se almacena un 1, mientras que en el resto de ellos un cero (ver tabla 7.16). Nótese que si se tuviera que escribir un 0 en la cinta, entonces en $\mathtt{D}'$ habría almacenado un cero; y que además si la máquina de Turing hubiera estado en condiciones de paro entonces el *flip-flop* `Hlt` almacenaría un 1.

En la tabla 7.16 se muestra las operaciones involucradas en el procesamiento de la información desde el momento en que llegan las señales $\mathtt{DB}_0$ y $\mathtt{DP}_0$ a la unidad de control hasta que salen las señales correspondientes al proceso de escritura ($\mathtt{W}_0$

y $\texttt{DF}_0$). Mientras que la tabla 7.17 muestra las operaciones desde que se libera el retardo-4 hasta que las señales para el movimiento de cabeza salen de la unidad de control. Este proceso se da por separado para asegurar que las señales involucradas en el proceso de escritura ($\texttt{W}_0$ y $\texttt{DF}_0$) lleguen primero a la i-ésima celda antes que las señales $\texttt{MB}_0$ y $\texttt{MF}_0$, las cuales indican que tipo de movimiento realizará la cabeza.

Unidad de control			
	Orden	Señal/ Componente	Valor
Entradas		Start	0
		$\texttt{DB}_0$	1
		$\texttt{DP}_0$	1
Valores internos		$\texttt{q}_0$	1
		$\texttt{q}_1$	0
		$\texttt{q}_2$	0
Cambios internos	1	D	1
	2	Compuertas lógicas	
	3	Retardo-1	1
	4	$\texttt{q}_0'$	1
	5	$\texttt{q}_1'$	0
	6	$\texttt{q}_2'$	0
	7	MF	1
	8	D′	1
	9	Hlt	0
	10	Retardo-2	1
	11	$\texttt{q}_0$	1
	12	$\texttt{q}_1$	0
	13	$\texttt{q}_2$	0
	14	Retardo-3	1
	15	$\texttt{A}_4$	1
	16	$\texttt{A}_1$	1
Salidas		$\texttt{R}_0$	0
		$\texttt{W}_0$	1
		$\texttt{DF}_0$	1
		$\texttt{MB}_0$	0
		$\texttt{MF}_0$	0

Tabla 7.16: Señales y componentes involucrados en el procesamiento de información. Se muestra desde el momento que la señales $\texttt{DB}_0$ y $\texttt{DP}_0$ llegan a la unidad de control hasta que las señales de escritura salen hacia la cinta.

En la tabla 7.18 se muestran las operaciones involucradas en la liberación del

Unidad de control			
	Orden	Señal/ Componente	Valor
Entradas		Start	0
		DB_0	0
		DP_0	0
Valores internos		q_0	1
		q_1	0
		q_2	0
		q'_0	1
		q'_1	0
		q'_2	0
		MF	1
		D′	1
		Hlt	0
Cambios internos	1	Retardo-4	1
	2	A_2	0
	3	A_3	1
Salidas		R_0	0
		W_0	0
		DF_0	0
		MB_0	0
		MF_0	1

Tabla 7.17: Señales y componentes involucrados en el procesamiento de información. Se muestra desde el momento que se libera el retardo-4 hasta que las señales correspondientes al movimiento de cabeza salen hacia la cinta.

retardo-5, el cual envía la señal a las celdas de memoria de que nuevamente se ejecutará el proceso de lectura.

Unidad de control			
	Orden	Señal/ Componente	Valor
Entradas		Start	0
		DB_0	0
		DP_0	0
Valores internos		q_0	1
		q_1	0
		q_2	0
		q'_0	1
		q'_1	0
		q'_2	0
		MF	1
		D′	1
		Hlt	0
Cambios internos	1	Retardo-5	1
	2	O_1	1
Salidas		R_0	1
		W_0	0
		DF_0	0
		MB_0	0
		MF_0	0

Tabla 7.18: Señales y componentes involucrados en el procesamiento de información. Se muestra desde el momento que se libera el retardo-5 hasta que la señal R_0 sale hacia la cinta.

7.6.3. Escritura

Recuérdese que el proceso de escritura comienza cuando se activa la señal W_0 desde la unidad de control. Junto con ella sale la señal DF_0 para transferir el dato que se escribirá en la cinta, que en este caso un 1. Dado que la cabeza no está en la celda cero, entonces a su paso por ella, ésta sólo transfiere las señales W_0 y DF_0 hacia las señales de salida W_1 y DF_1, respectivamente, para enviarlas a la siguiente celda de memoria (ver tabla 7.19).

En la celda $i = 1$ (donde está la cabeza) se bloquea la señal de escritura hacia la siguiente celda y se procede a almacenar el dato en el *flip-flop* D (ver tabla 7.20).

Celda cero			
	Orden	Señal/ Componente	Valor
Entradas		R_0	0
		W_0	1
		DF_0	1
		MB_0	0
		MF_0	0
		HB_1	0
		DB_1	0
		DP_1	0
Valores internos		H	0
Cambios internos	1	A_7	1
	2	A_6	1
Salidas		R_1	0
		W_1	1
		DF_1	1
		MB_1	0
		MF_1	0
		HF_1	0
		DB_0	0
		DP_0	0

Tabla 7.19: Señales y componentes involucrados en la celda cero para cuando se está yendo hacia la celda 1 para escribir el dato transferido a través de la señal DF_0.

i-ésima celda			
	Orden	Señal/ Componente	Valor
Entradas		R_i	0
		W_i	1
		DF_i	1
		MB_i	0
		MF_i	0
		HF_i	0
		HB_{i+1}	0
		DB_{i+1}	0
		DP_{i+1}	0
Valores internos		H	1
		D	1
Cambios internos	1	A_{13}	0
	2	A_{12}	0
	3	A_{20}	1
	4	D	1
Salidas		R_{i+1}	0
		W_{i+1}	0
		DF_{i+1}	0
		MB_{i+1}	0
		MF_{i+1}	0
		HF_{i+1}	0
		HB_i	0
		DB_i	0
		DP_i	0

Tabla 7.20: Señales y componentes involucrados en la i-ésima celda (i = 1) para el proceso de escritura. En este caso se almacenará un 1 en el *flip-flop* D.

7.6.4. Movimiento de cabeza

Recuérdese que el proceso de movimiento de cabeza comienza cuando las señales MB_0 y MF_0 salen en conjunto de la unidad de control hacia la cinta, y que además éstas salen un tiempo después de las señales W_0 y DF_0 para asegurar que primero se realicen las modificaciones correspondientes al proceso de escritura en la i-ésima celda.

Para este ejemplo en particular, sólo MF_0 transfiere un 1 debido a que la cabeza se moverá a la derecha. Dado que la cabeza no está en la celda cero, entonces a su paso por ella, ésta sólo transfiere las señales MB_0 y MF_0 hacia las señales de salida MB_1 y MF_1, respectivamente, para enviarlas a la siguiente celda de memoria (ver tabla 7.21).

Celda cero			
	Orden	Señal/ Componente	Valor
Entradas		R_0	0
		W_0	0
		DF_0	0
		MB_0	0
		MF_0	1
		HB_1	0
		DB_1	0
		DP_1	0
Valores internos		H	0
Cambios internos	1	A_8	0
	2	A_9	1
Salidas		R_1	0
		W_1	0
		DF_1	0
		MB_1	0
		MF_1	1
		HF_1	0
		DB_0	0
		DP_0	0

Tabla 7.21: Señales y componentes involucrados en la celda cero para cuando se está por apagar la cabeza de la celda 1.

En la celda $i = 1$ se bloquean las señales de movimiento de cabeza hacia la siguiente celda y, dado que se requiere mover la cabeza hacia la derecha, la señal $\texttt{HF}_{\texttt{i+1}}$ es activada para enviarla a la celda $i+1$. Posteriormente en la i-ésima celda se almacena un 0 en el *flip-flop* `H` para apagar la señal de la cabeza (ver tabla 7.22) y en la celda $i+1$ se activa la cabeza almacenando un 1 en su correspondiente *flip-flop* `H` (ver tabla 7.23). Con estos pasos se simula el movimiento de la cabeza a la derecha.

Nótese que en caso de que la cabeza se tenga que mover hacia la izquierda, entonces se activará la señal $\texttt{HB}_{\texttt{i}}$ en la celda i y posteriormente en la celda $i-1$ se encenderá la señal de cabeza.

i-ésima celda			
	Orden	Señal/ Componente	Valor
Entradas		$\texttt{R}_\texttt{i}$	0
		$\texttt{W}_\texttt{i}$	0
		$\texttt{DF}_\texttt{i}$	0
		$\texttt{MB}_\texttt{i}$	0
		$\texttt{MF}_\texttt{i}$	1
		$\texttt{HF}_\texttt{i}$	0
		$\texttt{HB}_{\texttt{i+1}}$	0
		$\texttt{DB}_{\texttt{i+1}}$	0
		$\texttt{DP}_{\texttt{i+1}}$	0
Valores internos	1	H	1
		D	1
Cambios internos	1	$\texttt{A}_{16}$	1
	2	$\texttt{A}_{21}$	0
	3	$\texttt{O}_5$	1
	4	$\texttt{A}_{19}$	1
	5	Retardo-1	1
	6	H	0
Salidas		$\texttt{R}_{\texttt{i+1}}$	0
		$\texttt{W}_{\texttt{i+1}}$	0
		$\texttt{DF}_{\texttt{i+1}}$	0
		$\texttt{MB}_{\texttt{i+1}}$	0
		$\texttt{MF}_{\texttt{i+1}}$	0
		$\texttt{HF}_{\texttt{i+1}}$	1
		$\texttt{HB}_\texttt{i}$	0
		$\texttt{DB}_\texttt{i}$	0
		$\texttt{DP}_\texttt{i}$	0

Tabla 7.22: Señales y componentes involucrados en la i-ésima celda (i = 1) para apagar la señal del *flip-flop* H que indica que la cabeza está en ella.

i-ésima celda			
	Orden	Señal/ Componente	Valor
Entradas		R_i	0
		W_i	0
		DF_i	0
		MB_i	0
		MF_i	0
		HF_i	1
		HB_{i+1}	0
		DB_{i+1}	0
		DP_{i+1}	0
Valores internos		H	0
		D	0
Cambios internos	1	O_2	1
	2	H	1
Salidas		R_{i+1}	0
		W_{i+1}	0
		DF_{i+1}	0
		MB_{i+1}	0
		MF_{i+1}	0
		HF_{i+1}	0
		HB_i	0
		DB_i	0
		DP_i	0

Tabla 7.23: Señales y compuertas involucradas en la i-ésima celda ($i = 2$) para encender la señal del *flip-flop* H.

Para este entonces, un nuevo ciclo de la máquina de Turing ya se ha generado (específicamente, cuando se liberó nuevamente la señal de lectura en el último paso del proceso de cambio de estado), por lo que ahora se irá a leer el segundo símbolo de la cadena w para repetir nuevamente los siguientes procesos involucrados en la simulación de la ejecución de la máquina de Turing.

Capítulo 8

Conclusiones

El contenido de esta investigación mostró el primer resultado en la literatura sobre la indecibilidad del clan-comportamiento. Por supuesto, el problema original contempla a las gráficas simples finitas, y aquí se probó dicho resultado para las gráficas automáticas, incluso si éstas consideran restricciones añadidas como que sean cuasi-clan-Helly y que tengan grado acotado y a lo más un vértice dominado. Pese a la relajación de la clase de gráficas, el análisis implicó la inexpresibilidad en lenguaje de primer orden de la clan-convergencia para gráficas automáticas, aún si se considera la generalización de cuantificadores (como por ejemplo: $\exists^{\infty}$, $\exists^{(k,m)}$ y $\exists^{k-ram}$) y si se incluye a ciertos cuantificadores restringidos (monádicos) de la teoría de segundo orden.

El resultado de la inexpresibilidad fue mejorado ya que se demostró que la propiedad de la clan-divergencia también es inexpresable en lenguaje de primer orden para gráficas finitas. Ahora, lo que queda pendiente es saber si la propiedad se mantiene inexpresable en lenguaje monádico de segundo orden, así como sucedió con las gráficas automáticas. Además, esto abre nuevas líneas de investigación para el estudio de la decibilidad computacional del problema de decisión del K-comportamiento.

La prueba de la indecibilidad dio como consecuencia el primer avance sobre la simulación de componentes digitales mediante gráficas de clanes. Principalmente lo que se buscaba era encontrar un conjunto funcionalmente completo de compuertas lógicas, ya fuera el conjunto $\{\texttt{AND}, \texttt{NOT}\}$ o el conjunto $\{\texttt{OR}, \texttt{NOT}\}$. Esta búsqueda generó la prueba de la no existencia de la compuerta `NOT` bajo la primera codificación, misma que se realizó utilizando resultados previos de las gráficas de clanes, tal como la desmantelabilidad de una gráfica a otra. Ya que se requería de una compuerta `NOT` se propuso una segunda codificación donde ésta fuera posible. En esta nueva codificación, el diseño de la compuerta `AND` está formada por las compuertas `OR` y `AND` de la primera codificación.

El diseño propuesto de la compuerta `AND` en la primera codificación genera el comportamiento esperado pero además produce más vértices cuando se aplica el operador de clanes sobre la gráfica que la simula, razón por la cual fue llamada `AND`-

sucia. Entonces, suponiendo la existencia de la compuerta AND-limpia en la primera codificación sería posible el diseño de la compuerta AND en la segunda codificación.

Cabe mencionar que aunque la compuerta AND-sucia no se logró estabilizar, ésta mostró el indicio del diseño de la gráfica que genera el comportamiento esperado en la compuerta lógica AND, sólo que aún se debe de buscar la manera de reproducir ese comportamiento de manera aislada y lograr que se mantenga en más iteraciones del operador de clanes.

De igual manera, con la hipótesis de la existencia del conjunto completo {NOT, AND}, en la segunda codificación, se propuso una simulación de cualquier máquina de Turing estrictamente binaria mediante circuitos digitales. Esta propuesta surgió porque no se encontró en la literatura ninguna simulación completa y explícita de máquinas de Turing usando compuertas lógicas. El hecho de que se consideren sólo máquinas de Turing estrictamente binarias no implica que el resultado esté límitado debido a que éstas pueden simular a una máquina de Turing que contenga a cualquier alfabeto y además al espacio en blanco.

Apéndice A

Código para la construcción de los componentes

En esta sección se presentan los códigos en YAGS/GAP que permiten explorar y diseñar los componentes de la *primera codificación* del capítulo 6. Estos requieren del software GAP 4.5 [22] y YAGS 0.0.3 [9] para ser ejecutados.

Para el uso de los códigos se necesita la creación de dos archivos con extensión **.g**. Uno de esos archivos debe ser llamado **gadgetData.g** y el otro **gadgets.g** ya que posteriormente son invocados bajo esos nombres en el mismo código o en su compilación. El primer archivo almacenará el código de la sección A.1 y el segundo el de la sección A.2, los cuales involucrán respectivamente el diseño y exploración de los componentes. Estos archivos también pueden ser descargados del sitio web citado en [10].

En la sección A.3 se muestran los pasos para compilar y generar los dibujos de los componentes, y en la sección A.4 se muestra el código en GAP para la experimentación (si es que lo desea el lector) en la creación de nuevos componentes.

Con la finalidad de guiar al lector, el código de todas las secciones está descrito por comentarios que comienzan con el símbolo #.

A.1. Código de construcción

```
# Bloque de construccion basico:
gg1:=Graph( rec( Category := SimpleGraphs, Order := 6, Size := 10, Adjacencies := [ [ 2, 4 ], [ 1, 3,
4, 5 ], [ 2, 4, 5, 6 ],  [ 1, 2, 3, 5 ], [ 2, 3, 4, 6 ], [ 3, 5 ] ] ) );
SetCoordinates(gg1,[ [ -285, -234 ], [ -260, -259 ], [ -235, -284 ], [ -236, -235 ], [ -211, -260 ],
[ -186, -285 ] ]);


# Bloque de construccion basico sin vertices dominados:
gg2:=Graph( rec( Category := SimpleGraphs, Order := 12, Size := 20, Adjacencies := [ [ 2, 4, 7 ], [ 1,
3, 4, 5, 8 ], [ 2, 4, 5, 6, 9 ], [ 1, 2, 3, 5, 10 ], [ 2, 3, 4, 6, 11 ], [ 3, 5, 12 ], [ 1, 8 ], [ 2,
7, 9 ], [ 3, 8 ], [ 4, 11 ], [ 5, 10, 12 ], [ 6, 11 ] ] ) );
SetCoordinates(gg2,[ [ -250, -234 ], [ -225, -259 ], [ -200, -284 ], [ -201, -235 ], [ -176, -260 ],
[ -151, -285 ], [ -285, -234 ], [ -260, -259 ], [ -235, -284 ], [ -166, -235 ], [ -141, -260 ], [
-116, -285 ] ]);
```

```
 # Canal:
gg3:=Graph( rec( Category := SimpleGraphs, Order := 21, Size := 44, Adjacencies := [ [ 2, 4, 16 ], [
1, 3, 4, 5, 17 ], [ 2, 4, 5, 6, 18 ], [ 1, 2, 3, 5, 7 ], [ 2, 3, 4, 6, 7, 8 ], [ 3, 5, 7, 8, 9 ], [ 4,
5, 6, 8, 10 ], [ 5, 6, 7,9, 10, 11 ], [ 6, 8, 10, 11, 12 ], [ 7, 8, 9, 11, 13 ], [ 8, 9, 10, 12, 13,
14 ], [ 9, 11, 13, 14, 15 ], [ 10, 11, 12, 14, 19 ], [ 11, 12, 13, 15, 20 ], [ 12, 14, 21 ], [ 1, 17
], [ 2, 16, 18 ], [ 3, 17 ], [ 13, 20 ], [ 14, 19, 21 ], [ 15, 20 ] ] ) );
SetCoordinates(gg3,[ [ -250, -234 ], [ -225, -259 ], [ -200, -284 ], [ -200, -235 ], [ -175, -260 ],
[ -150, -285 ], [ -150, -235 ], [ -125, -260 ], [ -100, -285 ], [ -100, -235 ], [ -75, -260 ], [ -50,
-285 ], [ -50, -235 ], [ -25, -260 ], [ 0, -285 ], [ -285, -234 ], [ -260, -259 ], [ -235, -284 ], [
-15,-235 ], [ 10, -260 ], [ 35, -285 ] ]);


# Canal con foton:
gg3a0:=Graph( rec( Category := SimpleGraphs, Order := 22, Size := 50, Adjacencies := [ [ 2, 4, 16 ],
[ 1, 3, 4, 5, 17, 22 ], [ 2, 4, 5, 6, 18, 22 ], [ 1, 2, 3, 5, 7, 22 ], [ 2, 3, 4, 6, 7, 8, 22 ], [ 3,
5, 7, 8, 9, 22 ], [ 4, 5, 6, 8, 10, 22 ], [ 5, 6, 7, 9, 10, 11 ], [ 6, 8, 10, 11, 12 ], [ 7, 8, 9, 11,
13 ], [ 8, 9, 10, 12, 13, 14 ], [ 9, 11, 13, 14, 15 ], [ 10, 11, 12, 14, 19 ], [ 11, 12, 13, 15, 20 ],
[ 12, 14, 21 ], [ 1, 17 ], [ 2, 16, 18 ], [ 3, 17 ], [ 13, 20 ], [ 14, 19, 21 ], [ 15, 20 ], [ 2, 3,
4, 5, 6, 7 ] ] ) );
SetCoordinates(gg3a0,[ [ -250, -234 ], [ -225, -259 ], [ -200, -284 ], [ -200, -235 ], [ -175, -260 ],
[ -150, -285 ], [ -150, -235 ], [ -125, -260 ], [ -100, -285 ], [ -100, -235 ], [ -75, -260 ], [ -50,
-285 ], [ -50, -235 ], [ -25, -260 ], [ 0, -285 ], [ -285, -234 ], [ -260, -259 ], [ -235, -284 ], [
-15, -235 ], [ 10, -260 ], [ 35, -285 ], [ -194, -268 ] ]);


# Canal con foton, iteracion K^2:
gg3a1:=K2(gg3a0);


# Canal con foton, iteracion K^4:
gg3a2:=K2(gg3a1);


# Canal con foton, iteracion K^6:
gg3a3:=K2(gg3a2);


# Compuerta OR con valores cero en las entradas A y B:
gg4:=Graph( rec( Category := SimpleGraphs, Order := 48, Size := 113, Adjacencies := [ [ 2, 4, 40 ], [
1, 3, 4, 5, 41 ], [ 2, 4, 5, 6, 42 ], [ 1, 2, 3, 5, 7 ], [ 2, 3, 4, 6, 7, 8 ], [ 3, 5, 7, 8, 9 ], [ 4,
5, 6, 8, 10 ], [ 5, 6, 7, 9, 10, 11 ], [ 6, 8, 10, 11, 12 ], [ 7, 8, 9, 11, 13 ], [ 8, 9, 10, 12, 13,
14 ], [ 9, 11, 13, 14, 15 ], [ 10, 11, 12, 14, 16, 37, 38, 39 ], [ 11, 12, 13, 15, 16, 17, 38, 39 ],
[ 12, 14, 16, 17, 18, 39 ], [ 13, 14, 15, 17, 19 ], [ 14, 15, 16, 18, 19, 20 ], [ 15, 17, 19, 20, 21
], [ 16, 17, 18, 20, 22 ], [ 17, 18, 19, 21, 22, 23 ], [ 18, 20, 22, 23, 24 ], [ 19, 20, 21, 23, 25 ],
[ 20, 21, 22, 24, 25, 26 ], [ 21, 23, 25, 26, 27 ], [ 22, 23, 24, 26, 43 ], [ 23, 24, 25, 27, 44 ], [
24, 26, 45 ], [ 29, 31, 46 ], [ 28, 30, 31, 32, 47 ], [ 29, 31, 32, 33, 48 ], [ 28, 29, 30, 32, 34 ],
[ 29, 30, 31, 33, 34, 35 ], [ 30, 32, 34, 35, 36 ], [ 31, 32, 33, 35, 37 ], [ 32, 33, 34, 36, 37, 38
], [ 33, 35, 37, 38, 39 ], [ 13, 34, 35, 36, 38 ], [ 13, 14, 35, 36, 37, 39 ], [ 13, 14, 15, 36, 38 ],
[ 1, 41 ], [ 2, 40, 42 ], [ 3, 41 ], [ 25, 44 ], [ 26, 43, 45 ], [ 27, 44 ], [ 28, 47 ], [ 29, 46, 48
], [ 30, 47 ] ] ) );
SetCoordinates(gg4,[ [ -245, -235 ], [ -220, -260 ], [ -195, -285 ], [ -195, -234 ], [ -170, -259 ],
[ -145, -284 ], [ -145, -233 ], [ -120, -258 ], [ -95, -283 ], [ -101, -208 ], [ -76, -233 ], [ -51,
-258 ], [ -56, -190 ], [ -31, -215 ], [ -6, -240 ], [ -6, -192 ], [ 19, -217 ], [ 44, -242 ], [ 44,
-193 ], [ 69, -218 ], [ 94, -243 ], [ 94, -193 ], [ 119, -218 ], [ 144, -243 ], [ 144, -193 ], [ 169,
-218 ], [ 194, -243 ], [ -250, -160 ], [ -225, -185 ], [ -200, -210 ], [ -200, -160 ], [ -175, -185 ],
[ -150, -210 ], [ -151, -163 ], [ -126, -188 ], [ -101, -213 ], [ -102, -174 ], [ -77, -199 ], [ -52,
-224 ], [ -280, -235 ], [ -255, -260 ], [ -230, -285 ], [ 179, -193 ], [ 204, -218 ], [ 229, -243 ],
[ -285, -160 ], [ -260, -185 ], [ -235, -210 ] ]);


# Compuerta OR con valores 0 y 1 en las entradas A y B, respectivamente:
gg4a0:=Graph( rec( Category := SimpleGraphs, Order := 49, Size := 119, Adjacencies := [ [ 2, 4, 40 ],
[ 1, 3, 4, 5, 41, 49 ], [ 2, 4, 5, 6, 42, 49 ], [ 1, 2, 3, 5, 7, 49 ], [ 2, 3, 4, 6, 7, 8, 49 ], [ 3,
```

```
5, 7, 8, 9, 49 ], [ 4, 5, 6, 8, 10, 49 ], [ 5, 6, 7, 9, 10, 11 ], [ 6, 8, 10, 11, 12 ], [ 7, 8, 9, 11,
13 ], [ 8, 9, 10, 12, 13, 14 ], [ 9, 11, 13, 14, 15 ], [ 10, 11, 12, 14, 16, 37, 38, 39 ], [ 11, 12,
13, 15, 16, 17, 38, 39 ], [ 12, 14, 16, 17, 18, 39 ], [ 13, 14, 15, 17, 19 ], [ 14, 15, 16, 18, 19,
20 ], [ 15, 17, 19, 20, 21 ], [ 16, 17, 18, 20, 22 ], [ 17, 18, 19, 21, 22, 23 ], [ 18, 20, 22, 23,
24 ], [ 19, 20, 21, 23, 25 ], [ 20, 21, 22, 24, 25, 26 ], [ 21, 23, 25, 26, 27 ], [ 22, 23, 24, 26,
43 ], [ 23, 24, 25, 27, 44 ], [ 24, 26, 45 ], [ 29, 31, 46 ], [ 28, 30, 31, 32, 47 ], [ 29, 31, 32,
33, 48 ], [ 28, 29, 30, 32, 34 ], [ 29, 30, 31, 33, 34, 35 ], [ 30, 32, 34, 35, 36 ], [ 31, 32, 33,
35, 37 ], [ 32, 33, 34, 36, 37, 38 ], [ 33, 35, 37, 38, 39 ], [ 13, 34, 35, 36, 38 ], [ 13, 14, 35,
36, 37, 39 ], [ 13, 14, 15, 36, 38 ], [ 1, 41 ], [ 2, 40, 42 ], [ 3, 41 ], [ 25, 44 ], [ 26, 43, 45 ],
[ 27, 44 ], [ 28, 47 ], [ 29, 46, 48 ], [ 30, 47 ], [ 2, 3, 4, 5, 6, 7 ] ] ) );
SetCoordinates(gg4a0,[ [ -245, -235 ], [ -220, -260 ], [ -195, -285 ], [ -195, -234 ], [ -170, -259 ],
[ -145, -284 ], [ -145, -233 ], [ -120, -258 ], [ -95, -283 ], [ -101, -208 ], [ -76, -233 ], [ -51,
-258 ], [ -56, -190 ], [ -31, -215 ], [ -6, -240 ], [ -6, -192 ], [ 19, -217 ], [ 44, -242 ], [ 44,
-193 ], [ 69, -218 ], [ 94, -243 ], [ 94, -193 ], [ 119, -218 ], [ 144, -243 ], [ 144, -193 ], [ 169,
-218 ], [ 194, -243 ], [ -250, -160 ], [ -225, -185 ], [ -200, -210 ], [ -200, -160 ], [ -175, -185 ],
[ -150, -210 ], [ -151, -163 ], [ -126, -188 ], [ -101, -213 ], [ -102, -174 ], [ -77, -199 ], [ -52,
-224 ], [ -280, -235 ], [ -255, -260 ], [ -230, -285 ], [ 179, -193 ], [ 204, -218 ], [ 229, -243 ],
[ -285, -160 ], [ -260, -185 ], [ -235, -210 ], [ -189, -269 ] ]);


# Compuerta OR con valores 0 y 1 en las entradas A y B, respectivamente. Iteracion K^2:
gg4a1:=K2(gg4a0);


# Compuerta OR con valores 0 y 1 en las entradas A y B, respectivamente. Iteracion K^4:
gg4a2:=K2(gg4a1);


# Compuerta OR con valores 0 y 1 en las entradas A y B, respectivamente. Iteracion K^6:
gg4a3:=K2(gg4a2);


# Compuerta OR con valores 0 y 1 en las entradas A y B, respectivamente. Iteracion K^8:
gg4a4:=K2(gg4a3);


# Compuerta OR con valores 0 y 1 en las entradas A y B, respectivamente. Iteracion K^10:
gg4a5:=K2(gg4a4);


# Compuerta OR con valores 0 y 1 en las entradas A y B, respectivamente. Iteracion K^12:
gg4a6:=K2(gg4a5);


# Compuerta OR con valores 0 y 1 en las entradas A y B, respectivamente. Iteracion K^14:
gg4a7:=K2(gg4a6);


# Compuerta OR con valores 1 y 0 en las entradas A y B, respectivamente:
gg4b0:=Graph( rec( Category := SimpleGraphs, Order := 49, Size := 119, Adjacencies := [ [ 2, 4, 40 ],
[ 1, 3, 4, 5, 41 ], [ 2, 4, 5, 6, 42 ], [ 1, 2, 3, 5, 7 ], [ 2, 3, 4, 6, 7, 8 ], [ 3, 5, 7, 8, 9 ],
[ 4, 5, 6, 8, 10 ], [ 5, 6, 7, 9, 10, 11 ], [ 6, 8, 10, 11, 12 ], [ 7, 8, 9, 11, 13 ], [ 8, 9, 10, 12,
13, 14 ], [ 9, 11, 13, 14, 15 ], [ 10, 11, 12, 14, 16, 37, 38, 39 ], [ 11, 12, 13, 15, 16, 17, 38, 39
], [ 12, 14, 16, 17, 18, 39 ], [ 13, 14, 15, 17, 19 ], [ 14, 15, 16, 18, 19, 20 ], [ 15, 17, 19, 20,
21 ], [ 16, 17, 18, 20, 22 ], [ 17, 18, 19, 21, 22, 23 ], [ 18, 20, 22, 23, 24 ], [ 19, 20, 21, 23,
25], [ 20, 21, 22, 24, 25, 26 ], [ 21, 23, 25, 26, 27 ], [ 22, 23, 24, 26, 43 ], [ 23, 24, 25, 27, 44
],[ 24, 26, 45 ], [ 29, 31, 46 ], [ 28, 30, 31, 32, 47, 49 ], [ 29, 31, 32, 33, 48, 49 ], [ 28, 29,
30, 32, 34, 49 ], [ 29, 30, 31, 33, 34, 35, 49 ], [ 30, 32, 34, 35, 36, 49 ], [ 31, 32, 33, 35, 37,
49 ],[ 32, 33, 34, 36, 37, 38 ], [ 33, 35, 37, 38, 39 ], [ 13, 34, 35, 36, 38 ], [ 13, 14, 35, 36, 37,
39 ], [ 13, 14, 15, 36, 38 ], [ 1, 41 ], [ 2, 40, 42 ], [ 3, 41 ], [ 25, 44 ], [ 26, 43, 45 ], [ 27,
44 ], [ 28, 47 ], [ 29, 46, 48 ], [ 30, 47 ], [ 29, 30, 31, 32, 33, 34 ] ] ) );
SetCoordinates(gg4b0,[ [ -245, -235 ], [ -220, -260 ], [ -195, -285 ], [ -195, -234 ], [ -170, -259 ],
```

```
[ -145, -284 ], [ -145, -233 ], [ -120, -258 ], [ -95, -283 ], [ -101, -208 ], [ -76, -233 ], [ -51,
-258 ], [ -56, -190 ], [ -31, -215 ], [ -6, -240 ], [ -6, -192 ], [ 19, -217 ], [ 44, -242 ], [ 44,
-193 ], [ 69, -218 ], [ 94, -243 ], [ 94, -193 ], [ 119, -218 ], [ 144, -243 ], [ 144, -193 ], [ 169,
-218 ], [ 194, -243 ], [ -250, -160 ], [ -225, -185 ], [ -200, -210 ], [ -200, -160 ], [ -175, -185 ],
[ -150, -210 ], [ -151, -163 ], [ -126, -188 ], [ -101, -213 ], [ -102, -174 ], [ -77, -199 ], [ -52,
-224 ], [ -280, -235 ], [ -255, -260 ], [ -230, -285 ], [ 179, -193 ], [ 204, -218 ], [ 229, -243 ],
[ -285, -160 ], [ -260, -185 ], [ -235, -210 ], [ -192, -192 ] ]);


# Compuerta OR con valores 1 y 0 en las entradas A y B, respectivamente. Iteracion K^2:
gg4b1:=K2(gg4b0);


# Compuerta OR con valores 1 y 0 en las entradas A y B, respectivamente. Iteracion K^4:
gg4b2:=K2(gg4b1);


# Compuerta OR con valores 1 y 0 en las entradas A y B, respectivamente. Iteracion K^6:
gg4b3:=K2(gg4b2);


# Compuerta OR con valores 1 y 0 en las entradas A y B, respectivamente. Iteracion K^8:
gg4b4:=K2(gg4b3);


# Compuerta OR con valores 1 y 0 en las entradas A y B, respectivamente. Iteracion K^10:
gg4b5:=K2(gg4b4);


# Compuerta OR con valores 1 y 0 en las entradas A y B, respectivamente. Iteracion K^12:
gg4b6:=K2(gg4b5);


# Compuerta OR con valores 1 y 0 en las entradas A y B, respectivamente. Iteracion K^14:
gg4b7:=K2(gg4b6);


# Compuerta OR con valores 1 en las entradas A y B:
gg4c0:=Graph( rec( Category := SimpleGraphs, Order := 50, Size := 125, Adjacencies := [ [ 2, 4, 40 ],
[ 1, 3, 4, 5, 41, 49 ], [ 2, 4, 5, 6, 42, 49 ], [ 1, 2, 3, 5, 7, 49 ], [ 2, 3, 4, 6, 7, 8, 49 ], [ 3,
5, 7, 8, 9, 49 ], [ 4, 5, 6, 8, 10, 49 ], [ 5, 6, 7, 9, 10, 11 ], [ 6, 8, 10, 11, 12 ], [ 7, 8, 9, 11,
13 ], [ 8, 9, 10, 12, 13, 14 ], [ 9, 11, 13, 14, 15 ], [ 10, 11, 12, 14, 16, 37, 38, 39 ], [ 11, 12,
13, 15, 16, 17, 38, 39 ], [ 12, 14, 16, 17, 18, 39 ], [ 13, 14, 15, 17, 19 ], [ 14, 15, 16, 18, 19,
20 ], [ 15, 17, 19, 20, 21 ], [ 16, 17, 18, 20, 22 ], [ 17, 18, 19, 21, 22, 23 ], [ 18, 20, 22, 23,
24 ], [ 19, 20, 21, 23, 25 ], [ 20, 21, 22, 24, 25, 26 ], [ 21, 23, 25, 26, 27 ], [ 22, 23, 24, 26,
43 ], [ 23, 24, 25, 27, 44 ], [ 24, 26, 45 ], [ 29, 31, 46 ], [ 28, 30, 31, 32, 47, 50 ], [ 29, 31,
32, 33, 48, 50 ], [ 28, 29, 30, 32, 34, 50 ], [ 29, 30, 31, 33, 34, 35, 50 ], [ 30, 32, 34, 35, 36,
50 ],[ 31, 32, 33, 35, 37, 50 ], [ 32, 33, 34, 36, 37, 38 ], [ 33, 35, 37, 38, 39 ], [ 13, 34, 35, 36,
38 ], [ 13, 14, 35, 36, 37, 39 ], [ 13, 14, 15, 36, 38 ], [ 1, 41 ], [ 2, 40, 42 ], [ 3, 41 ], [ 25,
44 ], [ 26, 43, 45 ], [ 27, 44 ], [ 28, 47 ], [ 29, 46, 48 ], [ 30, 47 ], [ 2, 3, 4, 5, 6, 7 ], [ 29,
30, 31, 32, 33, 34 ] ] ) );
SetCoordinates(gg4c0,[ [ -245, -235 ], [ -220, -260 ], [ -195, -285 ], [ -195, -234 ], [ -170, -259 ],
[ -145, -284 ], [ -145, -233 ], [ -120, -258 ], [ -95, -283 ], [ -101, -208 ], [ -76, -233 ], [ -51,
-258 ], [ -56, -190 ], [ -31, -215 ], [ -6, -240 ], [ -6, -192 ], [ 19, -217 ], [ 44, -242 ], [ 44,
-193 ], [ 69, -218 ], [ 94, -243 ], [ 94, -193 ], [ 119, -218 ], [ 144, -243 ], [ 144, -193 ], [ 169,
-218 ], [ 194, -243 ], [ -250, -160 ], [ -225, -185 ], [ -200, -210 ], [ -200, -160 ], [ -175, -185 ],
[ -150, -210 ], [ -151, -163 ], [ -126, -188 ], [ -101, -213 ], [ -102, -174 ], [ -77, -199 ], [ -52,
-224 ], [ -280, -235 ], [ -255, -260 ], [ -230, -285 ], [ 179, -193 ], [ 204, -218 ], [ 229, -243 ],
[ -285, -160 ], [ -260, -185 ], [ -235, -210 ], [ -189, -269 ], [ -193, -194 ] ]);
```

```
# Compuerta OR con valores 1 en las entradas A y B. Iteracion K^2:
gg4c1:=K2(gg4c0);


# Compuerta OR con valores 1 en las entradas A y B. Iteracion K^4:
gg4c2:=K2(gg4c1);


# Compuerta OR con valores 1 en las entradas A y B. Iteracion K^6:
gg4c3:=K2(gg4c2);


# Compuerta OR con valores 1 en las entradas A y B. Iteracion K^8:
gg4c4:=K2(gg4c3);


# Compuerta OR con valores 1 en las entradas A y B. Iteracion K^10:
gg4c5:=K2(gg4c4);


# Compuerta OR con valores 1 en las entradas A y B. Iteracion K^12:
gg4c6:=K2(gg4c5);


# Compuerta OR con valores 1 en las entradas A y B. Iteracion K^14:
gg4c7:=K2(gg4c6);


# Divisor:
gg5:=Graph( rec( Category := SimpleGraphs, Order := 48, Size := 113, Adjacencies := [ [ 2, 4, 40 ], [
1, 3, 4, 5, 41 ], [ 2, 4, 5, 6, 42 ], [ 1, 2, 3, 5, 7 ], [ 2, 3, 4, 6, 7, 8 ], [ 3, 5, 7, 8, 9 ], [ 4,
5, 6, 8, 10 ], [ 5, 6, 7, 9, 10, 11 ], [ 6, 8, 10, 11, 12 ], [ 7, 8, 9, 11, 13 ], [ 8, 9, 10, 12, 13,
14 ], [ 9, 11, 13, 14, 15 ], [ 10, 11, 12, 14, 16, 28 ], [ 11, 12, 13, 15, 16, 17, 28, 29 ], [ 12, 14,
16, 17, 18, 28, 29, 30 ], [ 13, 14, 15, 17, 19 ], [ 14, 15, 16, 18, 19, 20 ], [ 15, 17, 19, 20, 21 ],
[ 16, 17, 18, 20, 22 ], [ 17, 18, 19, 21, 22, 23 ], [ 18, 20, 22, 23, 24 ], [ 19, 20, 21, 23, 25 ], [
20, 21, 22, 24, 25, 26 ], [ 21, 23, 25, 26, 27 ], [ 22, 23, 24, 26, 43 ], [ 23, 24, 25, 27, 44 ], [
24, 26, 45 ], [ 13, 14, 15, 29, 31 ], [ 14, 15, 28, 30, 31, 32 ], [ 15, 29, 31, 32, 33 ], [ 28, 29,
30, 32, 34 ], [ 29, 30, 31, 33, 34, 35 ], [ 30, 32, 34, 35, 36 ], [ 31, 32, 33, 35, 37 ], [ 32, 33,
34, 36, 37, 38 ], [ 33, 35, 37, 38, 39 ], [ 34, 35, 36, 38, 46 ], [ 35, 36, 37, 39, 47 ], [ 36, 38,
48 ], [ 1, 41 ], [ 2, 40, 42 ], [ 3, 41 ], [ 25, 44 ], [ 26, 43, 45 ], [ 27, 44 ], [ 37, 47 ], [ 38,
46, 48 ], [ 39, 47 ] ] ) );
SetCoordinates(gg5,[ [ -250, -204 ], [ -225, -229 ], [ -200, -254 ], [ -200, -204 ], [ -175, -229 ],
[ -150, -254 ], [ -150, -206 ], [ -125, -231 ], [ -100, -256 ], [ -100, -206 ], [ -75, -231 ], [ -50,
-256 ], [ -50, -207 ], [ -25, -232 ], [ 0, -257 ], [ -6, -185 ], [ 19, -210 ], [ 44, -235 ], [ 40,
-165 ], [ 65, -190 ], [ 90, -215 ], [ 90, -163 ], [ 115, -188 ], [ 140, -213 ], [ 140, -162 ], [ 165,
-187  ], [ 190, -212 ], [ -3, -223 ], [ 22, -248 ], [ 47, -273 ], [ 46, -232 ], [ 71, -257 ], [ 96,
-282 ], [ 95, -235 ], [ 120, -260 ], [ 145, -285 ], [ 145, -235 ], [ 170, -260 ], [ 195, -285 ], [
-285, -204 ], [ -260, -229 ], [ -235, -254 ], [ 175, -162 ], [ 200, -187 ], [ 225, -212 ], [ 180,
-235 ], [ 205, -260 ], [ 230, -285 ] ]);


# Divisor con foton:
gg5a0:=Graph( rec( Category := SimpleGraphs, Order := 49, Size := 119, Adjacencies := [ [ 2, 4, 40],
[ 1, 3, 4, 5, 41, 49 ], [ 2, 4, 5, 6, 42, 49 ], [ 1, 2, 3, 5, 7, 49 ], [ 2, 3, 4, 6, 7, 8, 49 ], [
3, 5, 7, 8, 9, 49 ], [ 4, 5, 6, 8, 10, 49 ], [ 5, 6, 7, 9, 10, 11 ], [ 6, 8, 10, 11, 12 ], [ 7, 8, 9,
11, 13 ], [ 8, 9, 10, 12, 13, 14 ], [ 9, 11, 13, 14, 15 ], [ 10, 11, 12, 14, 16, 28 ], [ 11, 12, 13,
15, 16, 17, 28, 29 ], [ 12, 14, 16, 17, 18, 28, 29, 30 ], [ 13, 14, 15, 17, 19 ], [ 14, 15, 16, 18,
19, 20 ], [ 15, 17, 19, 20, 21 ], [ 16, 17, 18, 20, 22 ], [ 17, 18, 19, 21, 22, 23 ], [ 18, 20, 22,
23, 24 ], [ 19, 20, 21, 23, 25 ], [ 20, 21, 22, 24, 25, 26 ], [ 21, 23, 25, 26, 27 ], [ 22, 23, 24,
26, 43 ], [ 23, 24, 25, 27, 44 ], [ 24, 26, 45 ], [ 13, 14, 15, 29, 31 ], [ 14, 15, 28, 30, 31, 32 ],
[ 15, 29, 31, 32, 33 ], [ 28, 29, 30, 32, 34 ], [ 29, 30, 31, 33, 34, 35 ], [ 30, 32, 34, 35, 36 ],
[ 31, 32, 33, 35, 37 ], [ 32, 33, 34, 36, 37, 38 ], [ 33, 35, 37, 38, 39 ], [ 34, 35, 36, 38, 46 ],
```

```
[ 35, 36, 37, 39, 47 ], [ 36, 38, 48 ], [ 1, 41 ], [ 2, 40, 42 ], [ 3, 41 ], [ 25, 44 ], [ 26, 43, 45
], [ 27, 44 ], [ 37, 47 ], [ 38, 46, 48 ], [ 39, 47 ], [ 2, 3, 4, 5, 6, 7 ] ] ) );
SetCoordinates(gg5a0,[ [ -250, -204 ], [ -225, -229 ], [ -200, -254 ], [ -200, -204 ], [ -175, -229 ],
[ -150, -254 ], [ -150, -206 ], [ -125, -231 ], [ -100, -256 ], [ -100, -206 ], [ -75, -231 ], [ -50,
-256 ], [ -50, -207 ], [ -25, -232 ], [ 0, -257 ], [ -6, -185 ], [ 19, -210 ], [ 44, -235 ], [ 40,
-165 ], [ 65, -190 ], [ 90, -215 ], [ 90, -163 ], [ 115, -188 ], [ 140, -213 ], [ 140, -162 ], [ 165,
-187 ], [ 190, -212 ], [ -3, -223 ], [ 22, -248 ], [ 47, -273 ], [ 46, -232 ], [ 71, -257 ], [ 96,
-282 ], [ 95, -235 ], [ 120, -260 ], [ 145, -285 ], [ 145, -235 ], [ 170, -260 ], [ 195, -285 ], [
-285, -204 ], [ -260, -229 ], [ -235, -254 ], [ 175, -162 ], [ 200, -187 ], [ 225, -212 ], [ 180,
-235 ], [ 205, -260 ], [ 230, -285 ], [ -193, -238 ] ]);


# Divisor con foton. Iteracion K^2:
gg5a1:=K2(gg5a0);


# Divisor con foton. Iteracion K^4:
gg5a2:=K2(gg5a1);


# Divisor con foton. Iteracion K^6:
gg5a3:=K2(gg5a2);


# Divisor con foton. Iteracion K^8:
gg5a4:=K2(gg5a3);


# Divisor con foton. Iteracion K^10:
gg5a5:=K2(gg5a4);


# Divisor con foton. Iteracion K^12:
gg5a6:=K2(gg5a5);


# Divisor con foton. Iteracion K^14:
gg5a7:=K2(gg5a6);


# Empalme:
gg6:=Graph( rec( Category := SimpleGraphs, Order := 45, Size := 111, Adjacencies := [ [ 2, 4, 37 ], [
1, 3, 4, 5, 38 ], [ 2, 4, 5, 6, 39 ], [ 1, 2, 3, 5, 7 ], [ 2, 3, 4, 6, 7, 8 ], [ 3, 5, 7, 8, 9 ], [ 4,
5, 6, 8, 10, 22 ], [ 5, 6, 7, 9, 10, 11, 22, 23 ], [ 6, 8, 10, 11, 12, 22, 23, 24 ], [ 7, 8, 9, 11,
13 ], [ 8, 9, 10, 12, 13, 14 ], [ 9, 11, 13, 14, 15 ], [ 10, 11, 12, 14, 16, 31, 32, 33 ], [ 11, 12,
13, 15, 16, 17, 32, 33 ], [ 12, 14, 16, 17, 18, 33 ], [ 13, 14, 15, 17, 19 ], [ 14, 15, 16, 18, 19, 20
], [ 15, 17, 19, 20, 21 ], [ 16, 17, 18, 20, 40 ], [ 17, 18, 19, 21, 41 ], [ 18, 20, 42 ], [ 7, 8, 9,
23, 34 ], [ 8, 9, 22, 24, 34, 35 ], [ 9, 23, 31, 32, 33, 34, 35, 36 ], [ 26, 28, 31, 34, 35, 36 ], [
25, 27, 28, 29, 35, 36 ], [ 26, 28, 29, 30, 36 ], [ 25, 26, 27, 29, 43 ], [ 26, 27, 28, 30, 44 ], [
27, 29, 45 ], [ 13, 24, 25, 32, 35 ], [ 13, 14, 24, 31, 33, 35 ], [ 13, 14, 15, 24, 32 ], [ 22, 23,
24, 25, 35 ], [ 23, 24, 25, 26, 31, 32, 34, 36 ], [ 24, 25, 26, 27, 35 ], [ 1, 38 ], [ 2, 37, 39 ], [
3, 38 ], [ 19, 41 ], [ 20, 40, 42 ], [ 21, 41 ], [ 28, 44 ], [ 29, 43, 45 ],[ 30, 44 ] ] ) );
SetCoordinates(gg6,[ [ -172, 218 ], [ -168, 223 ], [ -163, 227 ], [ -121, 152 ], [ -117, 157 ], [-112,
161 ], [ -67, 89 ], [ -62, 94 ], [ -58, 99 ], [ 13, 89 ], [ 17, 94 ], [ 22, 99 ], [ 95, 87 ], [ 100,
91], [ 104, 96 ], [ 146, 152 ], [ 151, 157 ], [ 155, 161 ], [ 198, 217 ], [ 202, 222 ], [ 207, 226 ],
[ -24, 19 ], [ -20, 23 ], [ -15, 28 ], [ 18, -132 ], [ 23, -127 ], [ 27, -123 ], [ 16, -215 ], [ 21,
-211 ], [ 25, -206 ], [ 66, 9 ], [ 71, 13 ], [ 75, 18 ], [ 17, -49 ], [ 22, -45 ], [ 26, -40 ], [
-199, 249 ], [ -194, 255 ], [ -189, 260 ], [ 215, 248 ], [ 220, 250 ], [ 225, 253 ], [18, -247 ], [
24, -243 ], [29, -240 ] ]);
```

```
# Empalme con foton en A:
gg6a0:=Graph( rec( Category := SimpleGraphs, Order := 46, Size := 117, Adjacencies := [ [ 2, 4, 37 ],
[ 1, 3, 4, 5, 38, 46 ], [ 2, 4, 5, 6, 39, 46 ], [ 1, 2, 3, 5, 7, 46 ], [ 2, 3, 4, 6, 7, 8, 46 ], [ 3,
5, 7, 8, 9, 46 ], [ 4, 5, 6, 8, 10, 22, 46 ], [ 5, 6, 7, 9, 10, 11, 22, 23 ], [ 6, 8, 10, 11, 12, 22,
23, 24 ], [ 7, 8, 9, 11, 13 ], [ 8, 9, 10, 12, 13, 14 ], [ 9, 11, 13, 14, 15 ], [ 10, 11, 12, 14, 16,
31, 32, 33 ], [ 11, 12, 13, 15, 16, 17, 32, 33 ], [ 12, 14, 16, 17, 18, 33 ], [ 13, 14, 15, 17, 19 ],
[ 14, 15, 16, 18, 19, 20 ], [ 15, 17, 19, 20, 21 ], [ 16, 17, 18, 20, 40 ], [ 17, 18, 19, 21, 41 ], [
18, 20, 42 ], [ 7, 8, 9, 23, 34 ], [ 8, 9, 22, 24, 34, 35 ], [ 9, 23, 31, 32, 33, 34, 35, 36 ], [ 26,
28, 31, 34, 35, 36 ], [ 25, 27, 28, 29, 35, 36 ], [ 26, 28, 29, 30, 36 ], [ 25, 26, 27, 29, 43 ], [
26, 27, 28, 30, 44 ], [ 27, 29, 45 ], [ 13, 24, 25, 32, 35 ], [ 13, 14, 24, 31, 33, 35 ], [ 13, 14,
15, 24, 32 ], [ 22, 23, 24, 25, 35 ], [ 23, 24, 25, 26, 31, 32, 34, 36 ], [ 24, 25, 26, 27, 35 ], [
1, 38 ], [ 2, 37, 39 ], [ 3, 38 ], [ 19, 41 ], [ 20, 40, 42 ], [ 21, 41 ], [ 28, 44 ], [ 29, 43, 45 ],
[ 30, 44 ], [ 2, 3, 4, 5, 6, 7 ] ] ) );
SetCoordinates(gg6a0,[ [ -172, 218 ], [ -168, 223 ], [ -163, 227 ], [ -121, 152 ], [ -117, 157 ], [
-112, 161 ], [ -67, 89 ], [ -62, 94 ], [ -58, 99 ], [ 13, 89 ], [ 17, 94 ], [ 22, 99 ], [ 95, 87 ],
[ 100, 91 ], [ 104, 96 ], [ 146, 152 ], [ 151, 157 ], [ 155, 161 ], [ 198, 217 ], [ 202, 222 ], [ 207,
226 ], [ -24, 19 ], [ -20, 23 ], [ -15, 28 ], [ 18, -132 ], [ 23, -127 ], [ 27, -123 ], [ 16, -215 ],
[ 21, -211 ], [ 25, -206 ], [ 66, 9 ], [ 71, 13 ], [ 75, 18 ], [ 17, -49 ], [ 22, -45 ], [ 26, -40 ],
[ -199, 249 ], [ -194, 255 ], [ -189, 260 ], [ 215, 248 ], [ 220, 250 ], [ 225, 253 ], [ 18, -247 ],
[ 24, -243 ], [ 29, -240 ], [ -124, 168 ] ]);


# Empalme con foton en A. Iteracion K^2:
gg6a1:=K2(gg6a0);


# Empalme con foton en A:. Iteracion K^4:
gg6a2:=K2(gg6a1);


# Empalme con foton en A:. Iteracion K^6:
gg6a3:=K2(gg6a2);


# Empalme con foton en A:. Iteracion K^8:
gg6a4:=K2(gg6a3);


# Empalme con foton en A:. Iteracion K^10:
gg6a5:=K2(gg6a4);


# Empalme con foton en B:
gg6b0:=Graph( rec( Category := SimpleGraphs, Order := 46, Size := 117, Adjacencies := [ [ 2, 4, 37 ],
[ 1, 3, 4, 5, 38 ], [ 2, 4, 5, 6, 39 ], [ 1, 2, 3, 5, 7 ], [ 2, 3, 4, 6, 7, 8 ], [ 3, 5, 7, 8, 9 ], [
4, 5, 6, 8, 10, 22 ], [ 5, 6, 7, 9, 10, 11, 22, 23 ], [ 6, 8, 10, 11, 12, 22, 23, 24 ], [ 7, 8, 9, 11,
13 ], [ 8, 9, 10, 12, 13, 14 ], [ 9, 11, 13, 14, 15 ], [ 10, 11, 12, 14, 16, 31, 32, 33 ], [ 11, 12,
13, 15, 16, 17, 32, 33 ], [ 12, 14, 16, 17, 18, 33, 46 ], [ 13, 14, 15, 17, 19, 46 ], [ 14, 15, 16,
18, 19, 20, 46 ], [ 15, 17, 19, 20, 21, 46 ], [ 16, 17, 18, 20, 40, 46 ], [ 17, 18, 19, 21, 41, 46 ],
[ 18, 20, 42 ], [ 7, 8, 9, 23, 34 ], [ 8, 9, 22, 24, 34, 35 ],   [ 9, 23, 31, 32, 33, 34, 35, 36 ], [
26, 28, 31, 34, 35, 36 ], [ 25, 27, 28, 29, 35, 36 ], [ 26, 28, 29, 30, 36 ], [ 25, 26, 27, 29, 43 ],
[ 26, 27, 28, 30, 44 ], [ 27, 29, 45 ], [ 13, 24, 25, 32, 35 ], [ 13, 14, 24, 31, 33, 35 ], [ 13, 14,
15, 24, 32 ], [ 22, 23, 24, 25, 35 ], [ 23, 24, 25, 26, 31, 32, 34, 36 ], [ 24, 25, 26, 27, 35 ], [ 1,
38 ], [ 2, 37, 39 ], [ 3, 38 ], [ 19, 41 ], [ 20, 40, 42 ], [ 21, 41 ], [ 28, 44 ], [ 29, 43, 45 ], [
30, 44 ], [ 15, 16, 17, 18, 19, 20 ] ] ) );
SetCoordinates(gg6b0,[ [ -172, 218 ], [ -168, 223 ], [ -163, 227 ], [ -121, 152 ], [ -117, 157 ], [
-112, 161 ], [ -67, 89 ], [ -62, 94 ], [ -58, 99 ], [ 13, 89 ], [ 17, 94 ], [ 22, 99 ], [ 95, 87 ], [
100, 91 ], [ 104, 96 ], [ 146, 152 ], [ 151, 157 ], [ 155, 161 ], [ 198, 217 ], [ 202, 222 ], [ 207,
226 ], [ -24, 19 ], [ -20, 23 ], [ -15, 28 ], [ 18, -132 ], [ 23, -127 ], [ 27, -123 ], [ 16, -215 ],
[ 21, -211 ], [ 25, -206 ], [ 66, 9 ], [ 71, 13 ], [ 75, 18 ], [ 17, -49 ], [ 22, -45 ], [ 26, -40 ],
[ -199, 249 ], [ -194, 255 ], [ -189, 260 ], [ 215, 248 ], [ 220, 250 ], [ 225, 253 ], [ 18, -247 ],
[ 24, -243 ], [ 29, -240 ], [ 159, 167 ] ]);
```

```
# Empalme con foton en B. Iteracion K^2:
gg6b1:=K2(gg6b0);


# Empalme con foton en B. Iteracion K^4:
gg6b2:=K2(gg6b1);


# Empalme con foton en B. Iteracion K^6:
gg6b3:=K2(gg6b2);


# Empalme con foton en B. Iteracion K^8:
gg64:=K2(gg6b3);


# Empalme con foton en B. Iteracion K^2:
gg6b5:=K2(gg6b4);


# Empalme con foton en C:
gg6c0:=Graph( rec( Category := SimpleGraphs, Order := 46, Size := 117, Adjacencies := [ [ 2, 4, 37 ],
[ 1, 3, 4, 5, 38 ], [ 2, 4, 5, 6, 39 ], [ 1, 2, 3, 5, 7 ], [ 2, 3, 4, 6, 7, 8 ], [ 3, 5, 7, 8, 9 ], [
4, 5, 6, 8, 10, 22 ], [ 5, 6, 7, 9, 10, 11, 22, 23 ], [ 6, 8, 10, 11, 12, 22, 23, 24 ], [ 7, 8, 9, 11,
13 ], [ 8, 9, 10, 12, 13, 14 ], [ 9, 11, 13, 14, 15 ], [ 10, 11, 12, 14, 16, 31, 32, 33 ], [ 11, 12,
13, 15, 16, 17, 32, 33 ], [ 12, 14, 16, 17, 18, 33 ], [ 13, 14, 15, 17, 19 ], [ 14, 15, 16, 18, 19,
20 ], [ 15, 17, 19, 20, 21 ], [ 16, 17, 18, 20, 40 ], [ 17, 18, 19, 21, 41 ], [ 18, 20, 42 ], [ 7, 8,
9, 23, 34 ], [ 8, 9, 22, 24, 34, 35 ], [ 9, 23, 31, 32, 33, 34, 35, 36 ], [ 26, 28, 31, 34, 35, 36,
46 ], [ 25, 27, 28, 29, 35, 36, 46 ], [ 26, 28, 29, 30, 36, 46 ], [ 25, 26, 27, 29, 43, 46 ], [ 26,
27, 28, 30, 44, 46 ], [ 27, 29, 45 ], [ 13, 24, 25, 32, 35 ], [ 13, 14, 24, 31, 33, 35 ], [ 13, 14,
15, 24, 32 ], [ 22, 23, 24, 25, 35 ], [ 23, 24, 25, 26, 31, 32, 34, 36 ], [ 24, 25, 26, 27, 35, 46 ],
[ 1, 38 ], [ 2, 37, 39 ], [ 3, 38 ], [ 19, 41 ], [ 20, 40, 42 ], [ 21, 41 ], [ 28, 44 ], [ 29, 43, 45
], [ 30, 44 ], [ 25, 26, 27, 28, 29, 36 ] ] ) );
SetCoordinates(gg6c0,[ [ -172, 218 ], [ -168, 223 ], [ -163, 227 ], [ -121, 152 ], [ -117, 157 ], [
-112, 161 ], [ -67, 89 ], [ -62, 94 ], [ -58, 99 ], [ 13, 89 ], [ 17, 94 ], [ 22, 99 ], [ 95, 87 ], [
100, 91 ], [ 104, 96 ], [ 146, 152 ], [ 151, 157 ], [ 155, 161 ], [ 198, 217 ], [ 202, 222 ], [ 207,
226 ], [ -24, 19 ], [ -20, 23 ], [ -15, 28 ], [ 18, -132 ], [ 23, -127 ], [ 27, -123 ], [ 16, -215 ],
[ 21, -211 ], [ 25, -206 ], [ 66, 9 ], [ 71, 13 ], [ 75, 18 ], [ 17, -49 ], [ 22, -45 ], [ 26, -40 ],
[ -199, 249 ], [ -194, 255 ], [ -189, 260 ], [ 215, 248 ], [ 220, 250 ], [ 225, 253 ], [ 18, -247 ],
[ 24, -243 ], [ 29, -240 ], [ 21, -141 ] ]);


# Empalme con foton en C. Iteracion K^2:
gg6c1:=K2(gg6c0);


# Empalme con foton en C. Iteracion K^4:
gg6c2:=K2(gg6c1);


# Empalme con foton en C. Iteracion K^6:
gg6c3:=K2(gg6c2);


# Empalme con foton en C. Iteracion K^8:
gg6c4:=K2(gg6c3);


# Empalme con foton en C. Iteracion K^10:
gg6c5:=K2(gg6c4);
```

```
# Compuerta AND-sucia con valores cero en las entradas A y B:
gg7a0:=Graph( rec( Category := SimpleGraphs, Order := 66, Size := 162, Adjacencies := [ [ 2, 4 ], [ 1,
3, 5 ], [ 2, 6 ], [ 1, 5, 7 ], [ 2, 4, 6, 7, 8 ], [ 3, 5, 7, 8, 9 ], [ 4, 5, 6, 8, 10 ], [ 5, 6, 7, 9,
10, 11 ], [ 6, 8, 10, 11, 12 ], [ 7, 8, 9, 11, 13 ], [ 8, 9, 10, 12, 13, 14 ], [ 9, 11, 13, 14, 15 ],
[ 10, 11, 12, 14, 16 ], [ 11, 12, 13, 15, 16, 17 ], [ 12, 14, 16, 17, 18, 48, 51 ], [ 13, 14, 15, 17,
19, 48, 49, 50 ], [ 14, 15, 16, 18, 19, 20, 51, 52 ],   [ 15, 17, 19, 20, 21, 49, 52 ], [ 16, 17, 18,
20, 22, 51 ], [ 17, 18, 19, 21, 22, 23 ], [ 18, 20, 22, 23, 24 ], [ 19, 20, 21, 23, 25 ], [ 20, 21,
22, 24, 25, 26 ],   [ 21, 23, 25, 26, 27 ], [ 22, 23, 24, 26, 28 ], [ 23, 24, 25, 27, 28, 29 ], [ 24,
26, 28, 29, 30 ], [ 25, 26, 27, 29, 31 ], [ 26, 27, 28, 30, 32 ], [ 27, 29, 33 ], [ 28, 32 ], [ 29,
31, 33 ], [ 30, 32 ], [ 35, 37 ], [ 34, 36, 38 ], [ 35, 39 ], [ 34, 38, 40 ], [ 35, 37, 39, 40, 41 ],
[ 36, 38, 40, 41, 42 ], [ 37, 38, 39, 41, 43 ], [ 38, 39, 40, 42, 43, 44 ], [ 39, 41, 43, 44, 45 ], [
40, 41, 42, 44, 46 ], [ 41, 42, 43, 45, 46, 47 ], [ 42, 44, 46, 47, 48 ], [ 43, 44, 45, 47, 49 ], [
44, 45, 46, 48, 49, 50 ], [ 15, 16, 45, 47, 49, 50, 51 ], [ 16, 18, 46, 47, 48, 50, 52 ], [ 16, 47,
48, 49, 51, 52, 53 ], [ 15, 17, 19, 48, 50, 52, 53, 54 ], [ 17, 18, 49, 50, 51, 53, 55 ], [ 50, 51,
52, 54, 55, 56 ], [ 51, 53, 55, 56, 57 ], [ 52, 53, 54, 56, 58 ], [ 53, 54, 55, 57, 58, 59 ], [ 54,
56, 58, 59, 60 ], [ 55, 56, 57, 59, 61 ], [ 56, 57, 58, 60, 61, 62 ], [ 57, 59, 61, 62, 63 ], [ 58,
59, 60, 62, 64 ], [ 59, 60, 61, 63, 65 ], [ 60, 62, 66 ], [ 61, 65 ], [ 62, 64, 66 ], [ 63, 65 ] ]) );
SetCoordinates(gg7a0,[ [ -285, -244 ], [ -264, -264 ], [ -243, -285 ], [ -243, -244 ], [ -222, -264],
[ -201, -285 ], [ -201, -244 ], [ -181, -264 ], [ -160, -285 ], [ -160, -244 ], [ -139, -264 ], [
-118, -285 ], [ -118, -244 ], [ -97, -264 ], [ -76, -285 ], [ -76, -244 ], [ -56, -264 ], [ -36, -285
], [ -36, -244 ], [ -15, -264 ], [ 6, -285 ], [ 6, -244 ], [ 27, -264 ], [ 48, -285 ], [ 48, -244 ],
[ 69, -264 ], [ 89, -285 ], [ 89, -244 ], [ 110, -264 ], [ 131, -285 ], [ 131, -244 ], [ 152, -264 ],
[ 173, -285 ], [ -285, -160 ], [ -264, -181 ], [ -243, -202 ], [ -243, -160 ], [ -222, -181 ], [ -201,
-202 ], [ -201, -160 ], [ -181, -181 ], [ -160, -202 ], [ -160, -160 ], [ -139, -181 ], [ -118, -202
], [ -118, -160 ], [ -97, -181 ], [ -76, -202 ], [ -76, -160 ], [ -56, -181 ], [ -36, -202 ], [ -36,
-160 ], [ -15, -181 ], [ 6, -202 ], [ 6, -160 ], [ 27, -181 ], [ 48, -202 ], [ 48, -160 ], [ 69, -181
], [ 89, -202 ], [ 89, -160 ], [ 110, -181 ], [ 131, -202 ], [ 131, -160 ], [ 152, -181 ], [ 173,
-202 ] ]);

# Compuerta AND-sucia con valores cero en las entradas A y B. Iteracion K^2:
gg7a1:=K2(gg7a0);


# Compuerta AND-sucia con valores cero en las entradas A y B. Iteracion K^4:
gg7a2:=K2(gg7a1);


# Compuerta AND-sucia con valores cero en las entradas A y B. Iteracion K^6:
gg7a3:=K2(gg7a2);


# Compuerta AND-sucia con valores cero en las entradas A y B. Iteracion K^8:
gg7a4:=K2(gg7a3);


# Compuerta AND-sucia con valores cero en las entradas A y B. Iteracion K^10:
gg7a5:=K2(gg7a4);


# Compuerta AND-sucia con valores cero en las entradas A y B. Iteracion K^12:
gg7a6:=K2(gg7a5);


# Compuerta AND-sucia con valores cero en las entradas A y B. Iteracion K^14:
gg7a7:=K2(gg7a6);


# Compuerta AND-sucia con valores 0 y 1 en las entradas A y B, respectivamente:
gg7b0:=Graph( rec( Category := SimpleGraphs, Order := 67, Size := 168, Adjacencies := [ [ 2, 4 ], [ 1,
3, 5 ], [ 2, 6 ], [ 1, 5, 7 ], [ 2, 4, 6, 7, 8 ], [ 3, 5, 7, 8, 9 ], [ 4, 5, 6, 8, 10 ], [ 5, 6, 7, 9,
10, 11, 67 ], [ 6, 8, 10, 11, 12, 67 ], [ 7, 8, 9, 11, 13, 67 ], [ 8, 9, 10, 12, 13, 14, 67 ], [ 9,
11, 13, 14, 15, 67 ], [ 10, 11, 12, 14, 16, 67 ], [ 11, 12, 13, 15, 16, 17 ], [ 12, 14, 16, 17, 18,
```

```
48, 51 ], [ 13, 14, 15, 17, 19, 48, 49, 50 ], [ 14, 15, 16, 18, 19, 20, 51, 52 ], [ 15, 17, 19, 20,
21, 49, 52 ], [ 16, 17, 18, 20, 22, 51 ], [ 17, 18, 19, 21, 22, 23 ], [ 18, 20, 22, 23, 24 ], [ 19,
20, 21, 23, 25 ], [ 20, 21, 22, 24, 25, 26 ], [ 21, 23, 25, 26, 27 ],[ 22, 23, 24, 26, 28 ], [ 23, 24,
25, 27, 28, 29 ], [ 24, 26, 28, 29, 30 ], [ 25, 26, 27, 29, 31 ], [ 26, 27, 28, 30, 32 ], [ 27, 29,
33 ], [ 28, 32 ],[ 29, 31, 33 ], [ 30, 32 ], [ 35, 37 ], [ 34, 36, 38 ], [ 35, 39 ], [ 34, 38, 40 ],
[ 35, 37, 39, 40, 41 ], [ 36, 38, 40, 41, 42 ], [ 37, 38, 39, 41, 43 ], [ 38, 39, 40, 42, 43, 44 ], [
39, 41, 43, 44, 45 ], [ 40, 41, 42, 44, 46 ], [ 41, 42, 43, 45, 46, 47 ], [ 42, 44, 46, 47, 48 ], [
43, 44, 45, 47, 49 ], [ 44, 45, 46, 48, 49, 50 ], [ 15, 16, 45, 47, 49, 50, 51 ], [ 16, 18, 46, 47,
48, 50, 52 ], [ 16, 47, 48, 49, 51, 52, 53 ], [ 15, 17, 19, 48, 50, 52, 53, 54 ], [ 17, 18, 49, 50,
51, 53, 55 ], [ 50, 51, 52, 54, 55, 56 ], [ 51, 53, 55, 56, 57 ], [ 52, 53, 54, 56, 58 ], [ 53, 54,
55, 57, 58, 59 ], [ 54, 56, 58, 59, 60 ], [ 55, 56, 57, 59, 61 ], [ 56, 57, 58, 60, 61, 62 ], [ 57,
59, 61, 62, 63 ], [ 58, 59, 60, 62, 64 ], [ 59, 60, 61, 63, 65 ], [ 60, 62, 66 ], [ 61, 65 ], [ 62,
64, 66 ], [ 63, 65 ], [ 8, 9, 10, 11, 12, 13 ] ] ) );
SetCoordinates(gg7b0,[ [ -285, -244 ], [ -264, -264 ], [ -243, -285 ], [ -243, -244 ], [ -222, -264 ],
[ -201, -285 ], [ -201, -244 ], [ -181, -264 ], [ -160, -285 ], [ -160, -244 ], [ -139, -264 ], [
-118, -285 ], [ -118, -244 ], [ -97, -264 ], [ -76, -285 ], [ -76, -244 ], [ -56, -264 ], [ -36, -285
], [ -36, -244 ], [ -15, -264 ], [ 6, -285 ], [ 6, -244 ], [ 27, -264 ], [ 48, -285 ], [ 48, -244 ],
[ 69, -264 ], [ 89, -285 ], [ 89, -244 ], [ 110, -264 ], [ 131, -285 ], [ 131, -244 ], [ 152, -264 ],
[ 173, -285 ], [ -285, -160 ], [ -264, -181 ], [ -243, -202 ], [ -243, -160 ], [ -222, -181 ], [ -201,
-202 ], [ -201, -160 ], [ -181, -181 ], [ -160, -202 ], [ -160, -160 ], [ -139, -181 ], [ -118, -202
], [ -118, -160 ], [ -97, -181 ], [ -76, -202 ], [ -76, -160 ], [ -56, -181 ], [ -36, -202 ], [ -36,
-160 ], [ -15, -181 ], [ 6, -202 ], [ 6, -160 ], [ 27, -181 ], [ 48, -202 ], [ 48, -160 ], [ 69, -181
], [ 89, -202 ], [ 89, -160 ], [ 110, -181 ], [ 131, -202 ], [ 131, -160 ], [ 152, -181 ], [ 173,
-202 ], [ -146, -264 ] ]);


# Compuerta AND-sucia con valores 0 y 1 en las entradas A y B, respectivamente. Iteracion K^2:
gg7b1:=K2(gg7b0);


# Compuerta AND-sucia con valores 0 y 1 en las entradas A y B, respectivamente. Iteracion K^4:
gg7b2:=K2(gg7b1);


# Compuerta AND-sucia con valores 0 y 1 en las entradas A y B, respectivamente. Iteracion K^6:
gg7b3:=K2(gg7b2);


# Compuerta AND-sucia con valores 0 y 1 en las entradas A y B, respectivamente. Iteracion K^8:
gg7b4:=K2(gg7b3);


# Compuerta AND-sucia con valores 0 y 1 en las entradas A y B, respectivamente. Iteracion K^10:
gg7b5:=K2(gg7b4);


# Compuerta AND-sucia con valores 0 y 1 en las entradas A y B, respectivamente. Iteracion K^12:
gg7b6:=K2(gg7b5);


# Compuerta AND-sucia con valores 0 y 1 en las entradas A y B, respectivamente. Iteracion K^14:
gg7b7:=K2(gg7b6);


# Compuerta AND-sucia con valores 1 y 0 en las entradas A y B, respectivamente:
gg7c0:=Graph( rec( Category := SimpleGraphs, Order := 67, Size := 168, Adjacencies := [ [ 2, 4 ], [
1, 3, 5 ], [ 2, 6 ], [ 1, 5, 7 ], [ 2, 4, 6, 7, 8 ], [ 3, 5, 7, 8, 9 ], [ 4, 5, 6, 8, 10 ], [ 5, 6,
7, 9, 10, 11 ], [ 6, 8, 10, 11, 12 ], [ 7, 8, 9, 11, 13 ], [ 8, 9, 10, 12, 13, 14 ], [ 9, 11, 13, 14,
15 ], [ 10, 11, 12, 14, 16 ], [ 11, 12, 13, 15, 16, 17 ], [ 12, 14, 16, 17, 18, 48, 51 ], [ 13, 14,
15, 17, 19, 48, 49, 50 ], [ 14, 15, 16, 18, 19, 20, 51, 52 ], [ 15, 17, 19, 20, 21, 49, 52 ], [ 16,
17, 18, 20, 22, 51 ], [ 17, 18, 19, 21, 22, 23 ], [ 18, 20, 22, 23, 24 ], [ 19, 20, 21, 23, 25 ], [
20, 21, 22, 24, 25, 26 ], [ 21, 23, 25, 26, 27 ], [ 22, 23, 24, 26, 28 ], [ 23, 24, 25, 27, 28, 29
```

```
], [ 24, 26, 28, 29, 30 ], [ 25, 26, 27, 29, 31 ], [ 26, 27, 28, 30, 32 ], [ 27, 29, 33 ], [ 28, 32
], [ 29, 31, 33 ], [ 30, 32 ], [ 35, 37 ], [ 34, 36, 38 ], [ 35, 39 ], [ 34, 38, 40 ], [ 35, 37, 39,
40, 41 ], [ 36, 38, 40, 41, 42 ], [ 37, 38, 39, 41, 43 ], [ 38, 39, 40, 42, 43, 44, 67 ], [ 39, 41,
43, 44, 45, 67 ], [ 40, 41, 42, 44, 46, 67 ], [ 41, 42, 43, 45, 46, 47, 67 ], [ 42, 44, 46, 47, 48,
67 ], [ 43, 44, 45, 47, 49, 67 ],[ 44, 45, 46, 48, 49, 50 ], [ 15, 16, 45, 47, 49, 50, 51 ], [ 16,
18, 46, 47, 48, 50, 52 ], [ 16, 47, 48, 49, 51, 52, 53 ], [ 15, 17, 19, 48, 50, 52, 53, 54 ], [ 17,
18, 49, 50, 51, 53, 55 ], [ 50, 51, 52, 54, 55, 56 ], [ 51, 53, 55, 56, 57 ], [ 52, 53, 54, 56, 58
], [ 53, 54, 55, 57, 58, 59 ], [ 54, 56, 58, 59, 60 ], [ 55, 56, 57, 59, 61 ], [ 56, 57, 58, 60, 61,
62 ], [ 57, 59, 61, 62, 63 ], [ 58, 59, 60, 62, 64 ], [ 59, 60, 61, 63, 65 ], [ 60, 62, 66 ], [ 61,
65 ], [ 62, 64, 66 ], [ 63, 65 ], [ 41, 42, 43, 44, 45, 46 ] ] ) );
SetCoordinates(gg7c0,[ [ -285, -244 ], [ -264, -264 ], [ -243, -285 ], [ -243, -244 ], [ -222, -264 ],
[ -201, -285 ], [ -201, -244 ], [ -181, -264 ], [ -160, -285 ], [ -160, -244 ], [ -139, -264 ], [
-118, -285 ], [ -118, -244 ], [ -97, -264 ], [ -76, -285 ], [ -76, -244 ], [ -56, -264 ], [ -36, -285
], [ -36, -244 ], [ -15, -264 ], [ 6, -285 ], [ 6, -244 ], [ 27, -264 ], [ 48, -285 ], [ 48, -244 ],
[ 69, -264 ], [ 89, -285 ], [ 89, -244 ], [ 110, -264 ], [ 131, -285 ], [ 131, -244 ], [ 152, -264 ],
[ 173, -285 ], [ -285, -160 ], [ -264, -181 ], [ -243, -202 ], [ -243, -160 ], [ -222, -181 ], [ -201,
-202 ], [ -201, -160 ], [ -181, -181 ], [ -160, -202 ], [ -160, -160 ], [ -139, -181 ], [ -118, -202
], [ -118, -160 ], [ -97, -181 ], [ -76, -202 ], [ -76, -160 ], [ -56, -181 ], [ -36, -202 ], [ -36,
-160 ], [ -15, -181 ], [ 6, -202 ], [ 6, -160 ], [ 27, -181 ], [ 48, -202 ], [ 48, -160 ], [ 69, -181
], [ 89, -202 ], [ 89, -160 ], [ 110, -181 ], [ 131, -202 ], [ 131, -160 ], [ 152, -181 ], [ 173,
-202 ], [ -146, -181 ] ]);


# Compuerta AND-sucia con valores 1 y 0 en las entradas A y B, respectivamente. Iteracion K^2:
gg7c1:=K2(gg7c0);


# Compuerta AND-sucia con valores 1 y 0 en las entradas A y B, respectivamente. Iteracion K^4:
gg7c2:=K2(gg7c1);


# Compuerta AND-sucia con valores 1 y 0 en las entradas A y B, respectivamente. Iteracion K^6:
gg7c3:=K2(gg7c2);


# Compuerta AND-sucia con valores 1 y 0 en las entradas A y B, respectivamente. Iteracion K^8:
gg7c4:=K2(gg7c3);


# Compuerta AND-sucia con valores 1 y 0 en las entradas A y B, respectivamente. Iteracion K^10:
gg7c5:=K2(gg7c4);


# Compuerta AND-sucia con valores 1 y 0 en las entradas A y B, respectivamente. Iteracion K^12:
gg7c6:=K2(gg7c5);


# Compuerta AND-sucia con valores 1 y 0 en las entradas A y B, respectivamente. Iteracion K^14:
gg7c7:=K2(gg7c6);


# Compuerta AND-sucia con valores 1 en las entradas A y B:
gg7d0:=Graph( rec( Category := SimpleGraphs, Order := 68, Size := 174, Adjacencies := [ [ 2, 4 ], [
1, 3, 5 ], [ 2, 6 ], [ 1, 5, 7 ], [ 2, 4, 6, 7, 8 ], [ 3, 5, 7, 8, 9 ], [ 4, 5, 6, 8, 10 ], [ 5, 6,
7, 9, 10, 11, 67 ], [ 6, 8, 10, 11, 12, 67 ], [ 7, 8, 9, 11, 13, 67 ], [ 8, 9, 10, 12, 13, 14, 67 ],
[ 9, 11, 13, 14, 15, 67 ], [ 10, 11, 12, 14, 16, 67 ], [ 11, 12, 13, 15, 16, 17 ], [ 12, 14, 16, 17,
18, 8, 51 ], [ 13, 14, 15, 17, 19, 48, 49, 50 ], [ 14, 15, 16, 18, 19, 20, 51, 52 ], [ 15, 17, 19,
20, 21, 49, 52 ], [ 16, 17, 18, 20, 22, 51 ], [ 17, 18, 19, 21, 22, 23 ], [ 18, 20, 22, 23, 24 ], [
19, 20, 21, 23, 25 ], [ 20, 21, 22, 24, 25, 26 ], [ 21, 23, 25, 26, 27 ], [ 22, 23, 24, 26, 28 ], [
23, 24, 25, 27, 28, 29 ], [ 24, 26, 28, 29, 30 ], [ 25, 26, 27, 29, 31 ], [ 26, 27, 28, 30, 32 ], [
27, 29, 33 ], [ 28, 32 ], [ 29, 31, 33 ], [ 30, 32 ], [ 35, 37 ], [ 34, 36, 38 ], [ 35, 39 ], [ 34,
38, 40 ], [ 35, 37, 39, 40, 41 ], [ 36, 38, 40, 41, 42 ], [ 37, 38, 39, 41, 43 ], [ 38, 39, 40, 42,
```

```
43, 44, 68 ], [ 39, 41, 43, 44, 45, 68 ], [ 40, 41, 42, 44, 46, 68 ], [ 41, 42, 43, 45, 46, 47, 68
], [ 42, 44, 46, 47, 48, 68 ], [ 43, 44, 45, 47, 49, 68 ], [ 44, 45, 46, 48, 49, 50 ], [ 15, 16, 45,
47, 49, 50, 51 ], [ 16, 18, 46, 47, 48, 50, 52 ], [ 16, 47, 48, 49, 51, 52, 53 ], [ 15, 17, 19, 48,
50, 52, 53, 54 ], [ 17, 18, 49, 50, 51, 53, 55 ], [ 50, 51, 52, 54, 55, 56 ], [ 51, 53, 55, 56, 57
], [ 52, 53, 54, 56, 58 ], [ 53, 54, 55, 57, 58, 59 ], [ 54, 56, 58, 59, 60 ], [ 55, 56, 57, 59, 61
], [ 56, 57, 58, 60, 61, 62 ], [ 57, 59, 61, 62, 63 ], [ 58, 59, 60, 62, 64 ], [ 59, 60, 61, 63, 65
], [ 60, 62, 66 ], [ 61, 65 ], [ 62, 64, 66 ], [ 63, 65 ], [ 8, 9, 10, 11, 12, 13 ], [ 41, 42, 43,
44, 45, 46 ] ] ) );
SetCoordinates(gg7d0,[ [ -285, -244 ], [ -264, -264 ], [ -243, -285 ], [ -243, -244 ], [ -222, -264 ],
[ -201, -285 ], [ -201, -244 ], [ -181, -264 ], [ -160, -285 ], [ -160, -244 ], [ -139, -264 ], [
-118, -285 ], [ -118, -244 ], [ -97, -264 ], [ -76, -285 ], [ -76, -244 ], [ -56, -264 ], [ -36, -285
], [ -36, -244 ], [ -15, -264 ], [ 6, -285 ], [ 6, -244 ], [ 27, -264 ], [ 48, -285 ], [ 48, -244 ],
[ 69, -264 ], [ 89, -285 ], [ 89, -244 ], [ 110, -264 ], [ 131, -285 ], [ 131, -244 ], [ 152, -264 ],
[ 173, -285 ], [ -285, -160 ], [ -264, -181 ], [ -243, -202 ], [ -243, -160 ], [ -222, -181 ], [ -201,
-202 ], [ -201, -160 ], [ -181, -181 ], [ -160, -202 ], [ -160, -160 ], [ -139, -181 ], [ -118, -202
], [ -118, -160 ], [ -97, -181 ], [ -76, -202 ], [ -76, -160 ], [ -56, -181 ], [ -36, -202 ], [ -36,
-160 ], [ -15, -181 ], [ 6, -202 ], [ 6, -160 ], [ 27, -181 ], [ 48, -202 ], [ 48, -160 ], [ 69, -181
], [ 89, -202 ], [ 89, -160 ], [ 110, -181 ], [ 131, -202 ], [ 131, -160 ], [ 152, -181 ], [ 173,
-202 ], [ -146, -264 ], [ -146, -181 ] ]);


# Compuerta AND-sucia con valores 1 en las entradas A y B. Iteracion K^2:
gg7d1:=K2(gg7d0);


# Compuerta AND-sucia con valores 1 en las entradas A y B. Iteracion K^4:
gg7d2:=K2(gg7d1);


# Compuerta AND-sucia con valores 1 en las entradas A y B. Iteracion K^6:
gg7d3:=K2(gg7d2);


# Compuerta AND-sucia con valores 1 en las entradas A y B. Iteracion K^8:
gg7d4:=K2(gg7d3);


# Compuerta AND-sucia con valores 1 en las entradas A y B. Iteracion K^10:
gg7d5:=K2(gg7d4);


# Compuerta AND-sucia con valores 1 en las entradas A y B. Iteracion K^12:
gg7d6:=K2(gg7d5);


# Compuerta AND-sucia con valores 1 en las entradas A y B. Iteracion K^14:
gg7d7:=K2(gg7d6);
```

A.2. Código de exploración

```
# Se carga el paquete YAGS.
RequirePackage("yags");

# Abreviacion del metodo CliqueGraph.
K:=CliqueGraph;

# Calcula el conjunto de estrellas de k2g. Donde k2g es la segunda grafica de clanes de g y una es-
# trella es un clan de clanes tal que la interseccion de todos sus elementos (los cuales son clanes)
# no es vacio.
```

```
 Stars:=function(g,k2g)
  local L,N;
  N:=List(Vertices(g),x->Union([x],Adjacency(g,x)));
  L:=Filtered(Vertices(k2g),x->Basement(g,k2g,x) in N);
  return L;
end;

 # Para cada estrella de k2g (vease abajo), calcula un centro. Donde k2g supone ser la segunda grafi-
 # ca de clanes de g. Un centro de una estrella es un vertice de g que pertenece a la interseccion de
 # los clanes que pertenecen a la estrella. Una estrella es un clan de clanes que tiene una interse-
 # ccion total no vacia.
 StarCenters:=function(g,k2g)
  local L,N;
  N:=List(Vertices(g),x->Union([x],Adjacency(g,x)));
  L:=List(Vertices(k2g),q->Position(N,Basement(g,k2g,q)));
  return L;
end;

 # Calcula el conjunto de corbatas de k2g. Donde k2g supone ser la segunda grafica de clanes de g.
 # Una corbata es un clan de clanes que no es una estrella. En las graficas que representan a los
 # componentes, las corbatas son consideradas como vertices nuevos y se dibujan en rojo.
 Neckties:=function(g,k2g)
  return Difference(Vertices(k2g),Stars(g,k2g));
 end;

 # Calcula la segunda iteracion de las graficas de clanes de una grafica g. Ademas, identifica las
 # estrellas en la nueva grafica y las coloca en las mismas coordenadas que en sus centros correspon-
 # dientes. Esto permite un dibujo agradable de las graficas iteradas de clanes, asumiendo que g tiene
 # un dibujo adecuado.
 K2:=function(g)
  local sc,k2g,stars,neckties,pos1,pos2,q,x,N;
  k2g:=K(K(g));
  sc:=StarCenters(g,k2g);
  neckties:=Filtered(Vertices(k2g),q->sc[q]=fail);
  stars:=Filtered(Vertices(k2g),q->sc[q]<>fail);
  pos1:=Coordinates(g);pos2:=[];
  for q in stars do
    x:=sc[q];
    pos2[q]:=pos1[x];
  od;
  for q in neckties do
    N:=Intersection(Adjacency(k2g,q),stars);
    if N=[] then
      pos2[q]:=[0,0];
    else
      pos2[q]:=List(Sum(List(N,p->pos2[p]))/Length(N),Int);
    fi;
  od;
  SetCoordinates(k2g,pos2);
  return k2g;
end;

 # Se carga el codigo de la construccion de los componentes.
 Read("gadgetData.g");
```

A.3. Compilación

Un inicio de sesión típica en GAP, usando los archivos creados anteriormente, luce de la siguiente manera:

```
> gap
--- some GAP info here ---
gap> RequirePackage("yags");
Loading  YAGS - Yet Another Graph System 0.0.3.
Copyright (C) 2016 by the YAGS authors; for details type: ?yags:authors
This is free software under GPLv3; for details type: ?yags:copyright
true
gap> Read("gadgets.g");
gap> Draw(gg4c3,Neckties(gg4c2,gg4c3));
```

En esta ejecución se lee el archivo **gadget.g**, el cual a su vez carga al archivo **gadgetData.g**. Esto sucede sin problemas porque ambos archivos están almacenados en el mismo directorio que el directorio de trabajo de GAP.

La última línea de la ejecución anterior dibuja a la gráfica $gg4c3$ y además resalta en color rojo a los fotones de la misma. Los fotones de $gg4c3$ son recuperados con la función *Neckties* utilizando como parámetros a $gg4c2$ y a $gg4c3$. Donde $gg4c3 := K2(gg4c2)$. Esta técnica de coloreado no funciona para las gráficas iniciales, específicamente para las nombradas $gg??0$. Para ellas se considera que los fotones son representados por los últimos vértices agregados a la gráfica original (es decir, aquella que no tiene ningún fotón) y por lo tanto, éstos son los que deben ser pintados de color rojo. Por ejemplo, para colorear los fotones de la gráfica $gg4a0$ se ejecutan las siguientes instrucciones:

```
gap> Order(gg4);
48
gap> Order(gg4a0);
49
gap> Draw(gg4a0,[49]);
```

En este código se recupera el orden de la gráficas $gg4$ (gráfica original) y $gg4a0$. Esto indica que el vértice 49 es el fóton de $gg4a0$, mismo que se pintará de color rojo, tal como se indica en el segundo parámetro de la función $Draw$. De manera similiar se procede a pintar los fotones de la gráfica $gg4c0$:

```
gap> Order(gg4);
48
gap> Order(gg4c0);
50
gap> Draw(gg4c0,[49,50]);
```

A.4. Código para la construcción de nuevos componentes

```
# Esto es util para exportar descripciones legibles de YAGS para gráficas.
outfile:="outdata.g";
Export:=function(name,g)
   AppendTo(outfile,name,":=",g,";\n");
   AppendTo(outfile,"SetCoordinates(",name,",",Coordinates(g),");\n\n");
end;

# En las graficas que se tienen de ejemplo, un foton siempre es un nuevo vertice, el cual "casi" es
```

```
# un gemelo del vertice 'x', excepto que éste no es adyacente a 'y'.
AddFoton:=function(g,x,y)
 local X;
 X:=Union([x],Adjacency(g,x));
 X:=Difference(X,[y]);
 return AddVerticesByAdjacencies(g,[X]);
end;

# Esta funcion simplemente mueve las coordenadas de una grafica para colocarla en la esquina superi-
# or izquierda de la pantalla de visualizacion.
TopLeft:=function(g)
 local deltax, deltay,cc;
 cc:=Coordinates(g);
 deltax:= -Minimum(List(cc,z->z[1]))-285;
 deltay:= -Minimum(List(cc,z->z[2]))-285;
 SetCoordinates(g,cc+[deltax,deltay]);
end;

# Incrementos utilizados en la función 'MountingTracks'.
dx:=25;dy:=-25;dl:=35;

# Dada una digrafica D, esta funcion monta vias a las flechas de D. Por ejemplo, para construir
# el canal del componente gg3 se pueden ejecutar los siguientes comandos:
#
# gap> D:=PathGraph(5:GraphCategory:=OrientedGraphs);
# gap> Draw(D);
# gap> G:=MountTracks(D);
#
# Las coordenadas de D se utilizan para intentar obtener un dibujo bueno de G.
MountTracks:=function(D)
 local fab,n,eds,Eds,t,a,b,e,x,y,T,G,coords1,coords;
 fab:=[[1,1],[2,1],[2,2],[3,1],[3,2],[3,3]];
 n:=Order(D);
 T:=Filtered(Vertices(D),x->Length(InNeigh(D,x))<1 or
                            Length(Adjacency(D,x))< 1);
 t:=Length(T); # number of terminal vertices.

 Eds:=[];
 for x in [0..n+t-1] do
   Append(Eds,[[3*x+1,3*x+2],[3*x+2,3*x+3]]);
 od;
 for e in Edges(D) do
    a:=e[1];b:=e[2];
    Append(Eds,[3*(a-1),3*(b-1)]+fab);
 od;
 y:=3*n+1;
 for x in T do
    Append(Eds,[[3*(x-1)+1,y],[3*(x-1)+2,y+1],[3*(x-1)+3,y+2]]);
    y:=y+3;
 od;
 G:=GraphByEdges(Eds);
 coords:=Coordinates(D);
 coords:=Concatenation(List(coords,c->[c-[dx,dy],c,c+[dx,dy]]));
 Append(coords,List([1..3*t],function(z)
      local i,j;
      i:=Int((z+2)/3);
      j:=z-3*(i-1);
      if Length(InNeigh(D,T[i]))< 1 then
         return coords[3*(T[i]-1)+j]-[dl,0];
      else
         return coords[3*(T[i]-1)+j]+[dl,0];
      fi;
      end
  ));
```

```
 SetCoordinates(G,coords);
 return G;
end;
```

Bibliografía

[1] B.D. Acharya. Some queries on the periodicity and convergence of a graph. In *Professor P. L. Bhatnagar commemoration volume*, pages 185–205. Nat. Acad. Sci., Allahabad, 1979.

[2] V. Bárány. *Automatic Presentations of Infinite Structures*. PhD thesis, RWTH Aachen University, Budapest, Hungary, 2007. `https://logic.rwth-aachen.de/~vbarany/diss.pdf`.

[3] R. Berger. *The undecidability of the domino problem*. Memoirs of the American Mathematical Society **66** (1966) .

[4] A. Blumensath and E. Grädel. *Finite presentations of infinite structures: automata and interpretations*. Theory Comput. Syst. **37** (2004) 641–674. `10.1007/s00224-004-1133-y`.

[5] A. Bondy, G. Durán, M.C. Lin and J.L. Szwarcfiter. *A sufficient condition for self-clique graphs*. Electronic Notes on Discrete Mathematics **7** (2001) 19–23.

[6] J.A. Bondy and U.S.R. Murty. *Graph theory*, volume 244 of *Graduate Texts in Mathematics*. Springer, New York, 2008.

[7] C.F. Bornstein and J.L. Szwarcfiter. *On clique convergent graphs*. Graphs Combin. **11** (1995) 213–220.

[8] J. Britton. *The word problem*. Annals of Mathematics, second series **77** (1963) 16–32.

[9] C. Cedillo, R. MacKinney-Romero, M.A. Pizaña, I.A. Robles and R. Villarroel-Flores. *YAGS - Yet Another Graph System, Version 0.0.3*, 2016. `http://xamanek.izt.uam.mx/yags/`.

[10] C. Cedillo and M. Pizaña. *Simulating digital circuits with clique graphs (supplementary media)*. Available (2019) at : `http://xamanek.izt.uam.mx/map/gadgets/`.

[11] C. Cedillo and M.A. Pizaña. *Clique-convergence is undecidable for automatic graphs.* Journal of Graph Theory **96** (2021) 414–437. https://doi.org/10.1002/jgt.22622.

[12] C. Cedillo and M. Pizaña. *Simulating digital circuits with clique graphs.* Matemática Contemporânea **46** (2019) 185–193.

[13] C. Cedillo and M. Pizaña. *Clique-divergence is not first-order expressible for the class of finite graphs.* Ars Combinatoria **150** (2020) . To appear.

[14] B. Chazelle and L. Monier. *Unbounded hardware is equivalent to deterministic turing machines.* Theoretical Computer Science **24** (1983) 123–130. https://doi.org/10.1016/0304-3975(83)90044-0.

[15] F.F. Dragan. *Centers of graphs and the Helly property (in Russian).* PhD thesis, Moldava State University, Chisinău, Moldava, 1989.

[16] H.B. Enderton. *A mathematical introduction to logic.* Harcourt/Academic Press, Burlington, MA, second edition, 2001.

[17] F. Escalante. *Über iterierte Clique-Graphen.* Abh. Math. Sem. Univ. Hamburg **39** (1973) 59–68.

[18] F. Escalante. *Schnittverbände in Graphen.* Abh. Math. Sem. Univ. Hamburg **38** (1972) 199–220.

[19] M.E. Frías-Armenta. *Gráficas iteradas de clanes.* PhD thesis, Universidad Nacional Autónoma de México, 2000.

[20] M.E. Frías-Armenta, V. Neumann-Lara and M.A. Pizaña. *Dismantlings and iterated clique graphs.* Discrete Math. **282** (2004) 263–265.

[21] P. Galinier, M. Habib and C. Paul. Chordal graphs and their clique graphs. In *Graph-theoretic concepts in computer science (Aachen, 1995)*, pages 358–371. Springer, Berlin, 1995.

[22] The GAP Group. *GAP – Groups, Algorithms, and Programming, Version 4.3*, 2002. (http://www.gap-system.org).

[23] F. Harary. *Graph theory.* Addison-Wesley Publishing Co., Reading, Mass.-Menlo Park, Calif.-London, 1969.

[24] S.T. Hedetniemi and P.J. Slater. *Line graphs of triangleless graphs and iterated clique graphs.* Springer Lecture Notes in Math. **303** (1972) 139–147.

[25] S.T. Hedetniemi and P.J. Slater. Line graphs of triangleless graphs and iterated clique graphs. In *Graph theory and applications (Proc. Conf., Western Michigan Univ., Kalamazoo, Mich., 1972; dedicated to the memory of J. W. T. Youngs)*, pages 139–147. Lecture Notes in Math., Vol. 303. Springer, Berlin, 1972.

[26] B. Hedman. *Clique graphs of time graphs.* J. Combin. Theory Ser. B **37** (1984) 270–278.

[27] J.E. Hopcroft and J.D. Ullman. *Introduction to automata theory, languages, and computation.* Addison-Wesley Publishing Co., Reading, Mass., 1979. Addison-Wesley Series in Computer Science.

[28] N. Immerman. *Upper and lower bounds for first order expressibility.* J. Comput. System Sci. **25** (1982) 76–98. https://doi.org/10.1016/0022-0000(82)90011-3.

[29] W. Imrich and S. Klavžar. *Product graphs.* Wiley-Interscience Series in Discrete Mathematics and Optimization. Wiley-Interscience, New York, 2000. Structure and recognition, With a foreword by Peter Winkler.

[30] B. Khoussainov and A. Nerode. Automatic presentations of structures. In *Logic and computational complexity (Indianapolis, IN, 1994)*, volume 960 of *Lecture Notes in Comput. Sci.*, pages 367–392. Springer, Berlin, 1995. http://dx.doi.org/10.1007/3-540-60178-3_93.

[31] D. Kuske. Theories of automatic structures and their complexity. In *Algebraic informatics*, volume 5725 of *Lecture Notes in Comput. Sci.*, pages 81–98. Springer, Berlin, 2009. http://dx.doi.org/10.1007/978-3-642-03564-7_5.

[32] F. Larrión, C.P. de Mello, A. Morgana, V. Neumann-Lara and M.A. Pizaña. *The clique operator on cographs and serial graphs.* Discrete Math. **282** (2004) 183–191.

[33] F. Larrión and V. Neumann-Lara. *A family of clique divergent graphs with linear growth.* Graphs Combin. **13** (1997) 263–266.

[34] F. Larrión and V. Neumann-Lara. *Clique divergent graphs with unbounded sequence of diameters.* Discrete Math. **197/198** (1999) 491–501.

[35] F. Larrión and V. Neumann-Lara. *Locally C_6 graphs are clique divergent.* Discrete Math. **215** (2000) 159–170.

[36] F. Larrión and V. Neumann-Lara. *On clique-divergent graphs with linear growth.* Discrete Math. **245** (2002) 139–153.

[37] F. Larrión, V. Neumann-Lara and M.A. Pizaña. *Clique divergent clockwork graphs and partial orders (extended abstract).* Electronic Notes on Discrete Mathematics **7** (2001) 143–146.

[38] F. Larrión, V. Neumann-Lara and M.A. Pizaña. *Whitney triangulations, local girth and iterated clique graphs.* Discrete Math. **258** (2002) 123–135.

[39] F. Larrión, V. Neumann-Lara and M.A. Pizaña. *Clique convergent surface triangulations.* Mat. Contemp. **25** (2003) 135–143.

[40] F. Larrión, V. Neumann-Lara and M.A. Pizaña. *Clique divergent clockwork graphs and partial orders.* Discrete Appl. Math. **141** (2004) 195–207.

[41] F. Larrión, V. Neumann-Lara and M.A. Pizaña. *On expansive graphs.* European J. Combin. **30** (2009) 372–379. http://dx.doi.org/10.1016/j.ejc.2008.05.005.

[42] F. Larrión, M.A. Pizaña and R. Villarroel-Flores. *On self-clique shoal graphs.* Discrete Applied Mathematics **205** (2016) 86 – 100. http://dx.doi.org/10.1016/j.dam.2016.01.013.

[43] L. Libkin. *Elements of finite model theory.* Texts in Theoretical Computer Science. An EATCS Series. Springer-Verlag, Berlin, 2004 https://doi.org/10.1007/978-3-662-07003-1.

[44] J. Meidanis. *The clique operator.* Available (2001) at : http://www.ic.unicamp.br/~meidanis/research/clique/.

[45] C. Miller. *Desicion problems for groups.* Algorithms and Classification in Combinatorial Group Theory **23** (1992) 1–59.

[46] M. Morris Mano. *Digital design.* Prentice-Hall, 1984.

[47] V. Neumann-Lara. On clique-divergent graphs. In *Problèmes combinatoires et théorie des graphes (Colloq. Internat. CNRS, Univ. Orsay, Orsay, 1976)*, volume 260 of *Colloq. Internat. CNRS*, pages 313–315. CNRS, Paris, 1978.

[48] V. Neumann-Lara. Clique divergence in graphs. In L. Lovász and V.T. Sós, editors, *Algebraic methods in graph theory, Coll. Math. Soc. János Bolyai, vol. 25 Szeged*, pages 563–569. North-Holland, Amsterdam, 1981.

[49] V. Neumann-Lara. *Clique divergence in graphs. some variations.* Pub. Prelim. Inst. Mat. U.N.A.M, México **224** (1991) 1–14.

[50] M.A. Pizaña. *The icosahedron is clique divergent.* Discrete Math. **262** (2003) 229–239.

[51] E. Prisner. *Convergence of iterated clique graphs.* Discrete Math. **103** (1992) 199–207.

[52] E. Prisner. *Graph dynamics*, volume 338 of *Pitman Research Notes in Mathematics Series*. Longman, Harlow, 1995.

[53] E. Prisner. *Graph dynamics.* Longman, Harlow, 1995.

[54] S. Rubin. *Automata presenting structures: a survey of the finite string case.* Bull. Symbolic Logic **14** (2008) 169–209. 10.2178/bsl/1208442827.

[55] M. Sipser. *Introduction to the theory of computation.* PWS Publishing Company, 20 Park Plaza, Boston, MA 02116-4324, 1997.

[56] J. Spencer. *The strange logic of random graphs*, volume 22 of *Algorithms and Combinatorics.* Springer-Verlag, Berlin, 2001.

[57] J.L. Szwarcfiter. A survey on clique graphs. In B.A. Reed and C. Linhares-Sales, editors, *Recent advances in algorithms and combinatorics*, volume 11 of *CMS Books Math./Ouvrages Math. SMC*, pages 109–136. Springer, New York, 2003.

[58] J.L. Szwarcfiter. *Recognizing clique-Helly graphs.* Ars Combin. **45** (1997) 29–32.

Índice alfabético

Adyacente, 5
Algoritmo, 30
Arista, 5
Ascendiente, 33
Asignación de variables, 16
Autómata
  finito, 11
  linealmente acotado, 30
  finito síncrono, 18

Camino, 6
  longitud, 6
Clan, 9
  comportamiento, 9
  convergente, 9
  de clanes, 10
  divergente, 9
Clase elemental, 17
Configuración con estampa de tiempo, 33
  terminal, 45
Conjunto funcionalmente completo de compuertas, 61
Cono, 10
Consecuencia lógica, 17
Convolución, 18
Corbata, 10

Descendiente, 33
Descripción instantánea, 25
Diagrama de estados, 12
Digráfica, 8

Estrategia ganadora, 56
Estrella de un vértice, 10
Estructura, 16
  automática, 19
    automática
      dominio, 19
Exgrado, 9
Expresión regular, 13

Firma, 14
Flip-flop, 82
  SR, 83
Fotón, 63
Fórmula, 14, 15
Fórmula atómica, 15

Grado
  de un vértice, 5
Gráfica, 5
  a la larga $2 - self$, 52
  a la larga clan-Helly, 52
  cuasi-periódica, 80
  automática, 22
  completa, 7
  conexa, 7
  de clanes, 9
    iteración, 9
  desmantelable, 10
  dirigida, 8
  disconexa, 7
  discreta, 7
  finita, 7
  infinita, 8
  trayectoria, 7
  trivial, 7
  vacía, 7
Gráficas
  cuasi-isomorfas, 57
  cuasi-clan-Helly, 41

Ingrado, 9
Interpretación, 16
Invariante, 10
Isomorfismo, 7

Juego de Ehrenfeucht-Fraïssé de k-rondas, 55

Lenguaje
  recursivamente enumerable, 25
  recursivo, 25
  regular, 11
    cerradura, 13
    concatenación, 13
    unión, 13
Lógica de primer orden, 14
lógicamente válida, 17

Modelo, 16
  de teoría, 17
Modular, 72
Máquina de Turing, 24
  configuraciones, 25
Máquinas de Turing estrictamente binarias, 81

Operador de clanes, 9
Orden
  de una gráfica, 5

Perturbaciones viajeras, 47
Predecesor, 33
Presentación automática, 22
Problemas indecidibles, 30
Producto cartesiano, 18
Pulso, 87

Relación automática, 19

Satisfacibilidad, 17
Señal de control, 87
Señal de dato, 87
Señal local, 87
Subgráfica, 7
Sucesor, 33

Teoría
  consistente, 17
Triángulo extendido, 10
Turing-completo, 61
Término, 14, 15

Unidad de control, 24
Universo de discurso, 14

Variables libres, 15
Vecindad
  cerrada, 5
Vértice, 5
  dominado, 10
  incidente, 5

Ápice, 10

Casa abierta al tiempo
UNIVERSIDAD AUTÓNOMA METROPOLITANA

ACTA DE DISERTACIÓN PÚBLICA

No. 00014
Matrícula: 2143805389

Sobre la computabilidad del clan-comportamiento

Con base en la Legislación de la Universidad Autónoma Metropolitana, en la Ciudad de México se presentaron a las 10:00 horas del día 21 del mes de abril del año 2021 POR VÍA REMOTA ELECTRÓNICA, los suscritos miembros del jurado designado por la Comisión del Posgrado:

DR. JOSE DAVID FLORES PEÑALOZA
DR. RAFAEL VILLARROEL FLORES
DRA. MIKA OLSEN
DRA. MUCUY-KAK DEL CARMEN GUEVARA AGUIRRE
DR. BERNARDO LLANO PEREZ

Bajo la Presidencia del primero y con carácter de Secretario el último, se reunieron a la presentación de la Disertación Pública cuya denominación aparece al margen, para la obtención del grado de:

DOCTORA EN CIENCIAS (CIENCIAS Y TECNOLOGIAS DE LA INFORMACION)

DE: MARIA DEL CARMEN CEDILLO CHAGOYA

y de acuerdo con el artículo 78 fracción IV del Reglamento de Estudios Superiores de la Universidad Autónoma Metropolitana, los miembros del jurado resolvieron:

Aprobar

Acto continuo, el presidente del jurado comunicó a la interesada el resultado de la evaluación y, en caso aprobatorio, le fue tomada la protesta.

MARIA DEL CARMEN CEDILLO CHAGOYA
ALUMNA

REVISÓ
MTRA. ROSALIA SERRANO DE LA PAZ
DIRECTORA DE SISTEMAS ESCOLARES

DIRECTOR DE LA DIVISIÓN DE CBI
DR. JESUS ALBERTO OCHOA TAPIA

PRESIDENTE
DR. JOSE DAVID FLORES PEÑALOZA

VOCAL
DR. RAFAEL VILLARROEL FLORES

VOCAL
DRA. MIKA OLSEN

VOCAL
DRA. MUCUY-KAK DEL CARMEN GUEVARA AGUIRRE

SECRETARIO
DR. BERNARDO LLANO PEREZ

El presente documento cuenta con la firma –autógrafa, escaneada o digital, según corresponda- del funcionario universitario competente, que certifica que las firmas que aparecen en esta acta – Temporal, digital o dictamen- son auténticas y las mismas que usan los c.c. profesores mencionados en ella