Minería de procesos: algoritmos para el descubrimiento de procesos

Samuel López Perales

Trabajo de fin de grado de Matemáticas
Universidad de Zaragoza

Enero de 2025

Directora del trabajo: María Antonia Zapata Abad

Abstract

Process mining uses system-generated event logs to analyze how processes are executed in practice and to create models that represent these sequences of activities. This allows for the analysis, monitoring and improvement of real process within an organization. The models used are simplified and abstract representations of systems, that capture their essential aspects making them easier to understand and analyse. In this study, Petri nets are used as the main model to represent the dynamics of processes.

Process mining can be classified into three main types. The first is **process discovery**, which consist of automatically creating a model based on event logs. The second type is **conformance checking**, which compares the actual behavior recorded in logs with a predefined process model, allowing for the identification of activities that do not follow the expected pattern and marking the deviations. Finally, the third type is **optimization and enhancement**, which aims to identify areas for improvement in processes, such as eliminating inefficiencies, reorganizing sequences, or speeding up activities.

Moreover, process mining can be analyzed from different perspectives. Among the most relevant are the **control-flow perspective**, which examines the order and sequence of activities; the **organizational perspective**, which focuses on the resources involved, such as people and departments; and the **temporal perspective**, which analyzes the timing of events, idle times, and frequencies. In this study, we focus on the first type of process mining, process discovery, from the control-flow perspective.

Within the discovery algorithms, we describe two of them in this study: the α **algorithm** and the **inductive mining algorithm**. The α algorithm was one of the first to address concurrency in processes. Although it has limitations, such as the inability to accurately represent complex structures, its simplicity makes it a valuable tool for introducing the challenges of discovery. This algorithm identifies causal relationships between activities by detecting patterns in event logs. On the other hand, the **inductive mining algorithm** uses a divide-and-conquer strategy, processing logs in blocks and generating structured models in the form of process trees. One advantage of this algorithm is that it ensures the resulting models are correct from the beginning, overcoming issues related to incompleteness or errors in the data. Additionally, inductive discovery is particularly useful for large logs or logs with infrequent data, solving problems that the α algorithm cannot handle. Currently, inductive discovery is one of the leading approaches in process mining due to its flexibility, formal guarantees, and scalability.

Finally, using the process mining tool ProM, we have applied these two algorithms to an event log, which is publicly available and originates from a real traffic fine management system.

Resumen

La minería de procesos utiliza registros de eventos generados por sistemas de información para analizar cómo se ejecutan los procesos en la práctica y generar modelos que representen estas secuencias de actividades. Esto permite analizar, monitorizar y mejorar procesos reales dentro de una organización. Los modelos empleados son representaciones simplificadas y abstractas de sistemas que capturan los aspectos esenciales, facilitando su comprensión y análisis. En este trabajo, se usan redes de Petri como principal modelo para representar la dinámica de los procesos.

La minería de procesos se puede clasificar en tres tipos principales. El primero es el **descubrimiento de procesos**, que consiste en crear automáticamente un modelo a partir de registros de eventos. El segundo tipo es la **verificación de conformidad**, que compara el comportamiento real registrado en los logs con un modelo de proceso predefinido, permitiendo identificar actividades que no siguen el modelo esperado y marcando las desviaciones. Finalmente, el tercer tipo es la **optimización y mejora**, que busca identificar áreas de mejora en los procesos, como eliminar ineficiencias, reorganizar secuencias o acelerar actividades.

Además, la minería de procesos puede analizarse desde distintas perspectivas. Entre las más relevantes está la perspectiva de **control de flujo**, que analiza el orden y la secuencia de las actividades. También está la perspectiva **organizacional**, que examina los recursos involucrados, como personas y departamentos, y la perspectiva **temporal**, que analiza la ocurrencia temporal de eventos, tiempos muertos y frecuencias. En este trabajo nos centramos en el primer tipo de minería de procesos, el descubrimiento de procesos, desde la perspectiva de control de flujo.

Dentro de los algoritmos de descubrimiento, en este trabajo vamos a describir dos de ellos, el **algoritmo** α y el **algoritmo de minería inductiva**. El algoritmo α fue uno de los primeros en abordar la concurrencia en los procesos. Aunque presenta limitaciones, como la incapacidad de representar con precisión estructuras complejas, su simplicidad lo convierte en una herramienta valiosa para introducirse en los desafíos del descubrimiento. Este algoritmo identifica relaciones causales entre actividades al buscar patrones en los registros de eventos. Por otro lado, el **algoritmo de minería inductiva** emplea una estrategia de divide y vencerás, procesando registros en bloques y generando modelos estructurados en forma de árboles de procesos. Una de las ventajas de este algoritmo es que garantiza que los modelos resultantes sean correctos desde el principio, superando problemas de incompletitud o errores en los datos. Además, el descubrimiento inductivo es especialmente útil para registros grandes o con datos infrecuentes, resolviendo problemas que el algoritmo α no puede abordar. Actualmente, el descubrimiento inductivo es uno de los enfoques líderes en minería de procesos gracias a su flexibilidad, garantías formales y escalabilidad.

Finalmente, haciendo uso de la herramienta de minería de procesos ProM, hemos aplicado estos dos algoritmos a un registro de eventos, que está disponible públicamente, y que proviene de un sistema real de gestión de multas de tráfico.

Índice general

Capítulo 1

Introducción a la minería de procesos

1.1. Contextualización

Los sistemas de información registran continuamente lo que ocurre en los procesos mediante eventos almacenados en registros o logs. Estos registros contienen detalles sobre la secuencia de actividades que se van produciendo. El objetivo de la ***minería de procesos*** es extraer información de valor a partir de estos eventos, permitiendo analizar cómo se ejecutan los procesos en la práctica y generar modelos que representen estas secuencias de actividades, lo que facilita detectar desviaciones respecto a los modelos teóricos. Haciendo uso de estos registros, la minería de procesos permite analizar, monitorizar y mejorar los procesos reales dentro de una organización. Las **referencias** que hemos utilizado principalmente para desarrollar este capítulo han sido [1], [2] y [3].

Un concepto relacionado con la minería de procesos que queremos introducir previamente es el concepto de *modelo*, que se define como una representación simplificada y abstracta de un sistema. Su objetivo principal es capturar los aspectos esenciales y relevantes del sistema en cuestión, al tiempo que simplifica y omite detalles innecesarios para facilitar su comprensión y análisis. Al utilizarse como una herramienta para comprender y analizar sistemas complejos, es fundamental que el modelo sea más fácil de manejar que el sistema original. En este trabajo nos interesan sobre todo los modelos que representan la dinámica de un sistema, en concreto, utilizaremos las Redes de Petri.

La minería de procesos es una disciplina muy amplia que abarca una gran cantidad de técnicas, herramientas y métodos que pretenden extraer conocimiento de los registros de eventos. En concreto, se diferencian distintos *tipos* de minería de procesos en función de la tarea principal que se desea llevar a cabo. Por otro lado, de forma ortogonal, la minería de procesos puede tener distintas *perspectivas* sobre la misma realidad, dando lugar a modelos diferentes. A continuación, vamos a comentar los distintos tipos y perspectivas que aparecen en la minería de procesos, lo que nos permitirá contextualizar el contenido de este trabajo.

Los tres principales tipos de la minería de procesos son:

1. **Descubrimiento de procesos:** A partir de un registro de los eventos que se producen en un sistema (log), el descubrimiento de procesos consiste en crear automáticamente un modelo que describa el comportamiento que se ha observado.
   Los métodos utilizados para el descubrimiento de procesos son, principalmente, algoritmos que toman un registro de eventos y generan un modelo de procesos (por ejemplo, una *red de Petri*) que describe las posibles secuencias de actividades observadas.
   El descubrimiento permite identificar cómo se están llevando a cabo los procesos en la práctica, lo cual puede diferir de cómo se supone que deben ejecutarse según los modelos formales o la documentación interna de la empresa.

2. **Verificación de conformidad:** La verificación de conformidad consiste en comparar el comporta-

miento real registrado en los logs de eventos con un modelo de proceso predefinido o esperado. El objetivo aquí es identificar si el proceso real sigue el modelo teórico, y en caso contrario, detectar las diferencias o desviaciones.
Se toma el modelo de proceso teórico, como podría ser un diagrama de flujo de trabajo o una red de Petri, y se reproduce el registro de eventos sobre este modelo. Si hay actividades que no siguen el modelo, o si el orden de las mismas no coincide con lo esperado, se marcan como desviaciones. Esta verificación es especialmente útil para asegurar que los procesos cumplen con regulaciones, políticas internas o estándares de calidad. También permite identificar errores en la ejecución que podrían estar pasando inadvertidos.

3. **Optimización y mejora:** El objetivo es identificar áreas de mejora en los procesos. En concreto, se busca optimizar el rendimiento del proceso, eliminando ineficiencias, automatizando pasos redundantes o acelerando la ejecución de ciertas actividades.
Al analizar el modelo de procesos junto con los registros de eventos, es posible identificar actividades que están creando cuellos de botella, pasos innecesarios o secuencias que podrían ser reorganizadas para mejorar el flujo.
Esta fase no solo permite optimizar el proceso una vez, sino que fomenta un ciclo de mejora continua. Por ejemplo, una vez optimizado un proceso, el sistema puede seguir monitorizando los nuevos registros de eventos para asegurarse de que las mejoras implementadas están dando los resultados esperados y, si es necesario, realizar ajustes adicionales.

Tal como hemos comentado, en la minería de procesos, además, existen varias perspectivas para analizar los registros. Una de ellas es la **perspectiva de control de flujo** (¿cómo?), que se enfoca en el orden y la secuencia de las actividades dentro de un proceso, buscando representar todas las rutas posibles mediante modelos como redes de Petri. También están la **perspectiva organizacional** (¿quién?), que analiza los recursos implicados (personas, departamentos, etc.), la **perspectiva de casos** (¿qué?), que examina las características específicas de cada caso procesado, es decir, la instancia concreta que está siendo tratada (multa, factura, pedido, etc.), y la **perspectiva de tiempo** (¿cuándo?), que analiza aspectos relacionados con la ocurrencia temporal de los eventos (tiempos muertos, frecuencia, etc.).

Una vez vistos los tipos y perspectivas que abarca la minería de procesos, comentar que en este trabajo nos centraremos en el **descubrimiento de procesos** desde la **perspectiva de control de flujo**. En concreto, presentaremos dos algoritmos de descubrimiento, el **algoritmo α** y un **algoritmo inductivo**. En ambos casos, a partir de un registro de eventos se obtiene como modelo resultante una red de Petri. Por esta razón, antes de pasar a explicar estos algoritmos, vamos a explicar las redes de Petri (sección 1.2) y los registros de eventos (sección 1.3).

1.2. Red de Petri

Definición 1. *(Grafo dirigido bipartito)* Dado un grafo $G = (V,E)$, donde V es el conjunto de vértices del grafo, y E es el conjunto de aristas dirigidas que conectan pares ordenados de vértices en V, se dice que el grafo es *bipartito* si existe una partición del conjunto de vértices V en dos subconjuntos disjuntos U y W tal que:

- $U \cap W = \emptyset$ (los subconjuntos son disjuntos),
- $U \cup W = V$ (la unión de los subconjuntos es igual al conjunto de vértices V),
- Cada arista $e \in E$ conecta un vértice en U con un vértice en W o uno de W con uno de U. Es decir, para cualquier arista $(x_1, x_2) \in E$, se cumple que $x_1 \in U$ y $x_2 \in W$, o bien, $x_2 \in U$ y $x_1 \in W$

Nota: Un grafo bipartito se puede representar como $G = (U, W, E)$, donde U y W son los dos conjuntos de vértices y $E \subseteq (U \times W) \cup (W \times U)$ es el conjunto de aristas.

Una *red de Petri* es un grado dirigido bipartito cuyos vértices son lugares o transiciones. Los lugares se representan gráficamente mediante círculos y las transiciones por rectángulos, tal y como podemos ver en el ejemplo de red de Petri que aparece en la Figura 1.1. Esta red de Petri representa las distintas actividades que se pueden llevar a cabo para gestionar los pedidos que llegan a una empresa, desde que se recibe el pedido hasta que se factura o se anula. Como veremos a continuación, los lugares pueden contener marcas que pueden fluir a través de la red en función de una regla de activación.

Definición 2. *(Red de Petri)* Una *red de Petri* es una tupla $N = (P,T,F)$ con P el conjunto de *lugares*, T el conjunto de *transiciones*, $P \cap T = \emptyset$, y $F \subseteq (P \times T) \cup (T \times P)$ la *relación de flujo*.

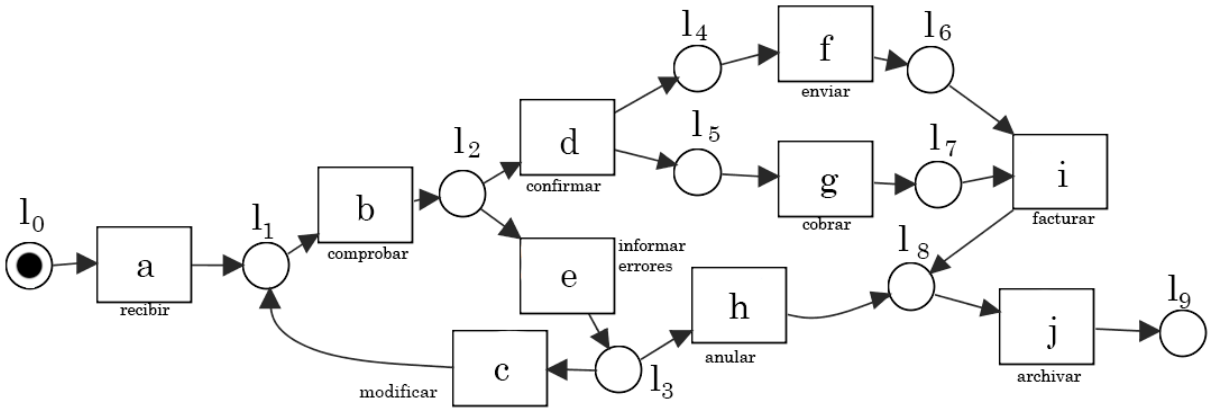

Figura 1.1: Ejemplo de una Red de Petri que representa una gestión de pedidos.

Para introducir el concepto de *marcado* de una red de petri, debemos explicar previamente el concepto de multiconjunto. Un *multiconjunto* es una generalizacion del concepto de conjunto en el que cada elemento puede aparecer varias veces. Dado un conjunto de elementos A, definimos $\mathscr{B}(A)$ como el conjunto de multiconjuntos cuyos elementos pertenecen a A. Por ejemplo, si $A = \{a,b\}$, un ejemplo de elemento en $\mathscr{B}(A)$ sería $[a,a,b]$, donde a aparece dos veces y b una vez. Análogamente a como se hace con los conjuntos, se define la suma de dos multiconjuntos ($X \uplus Y$), la diferencia ($X \setminus Y$), la presencia de un elemento en un multiconjunto ($x \in X$) y la noción de subconjunto ($X \leq Y$). Por ejemplo, si:

$$X = \{a,a,b\}, \quad Y = \{a,b,b,c\},$$

entonces:

$$X \uplus Y = \{a,a,a,b,b,b,c\},$$

y $X \leq Y$ no se cumple, porque el número de veces que aparece a en X es mayor al número de veces que aparece en Y, pero $X \leq X \uplus Y$ sí que se cumple.

Definición 3. *(Marcado)* Sea $N = (P,T,F)$ una red de Petri. Un *marcado* M es un multiconjunto de lugares, es decir, $M \in \mathscr{B}(P)$.

En una red de Petri, el marcado es un multiconjunto de lugares que indica cuántas marcas contiene cada lugar, es decir, nos indica la distribución de marcas en los lugares. Las marcas se representan gráficamente como puntos dentro de los lugares. En la Figura 1.1, el marcado consta de una única marca en el lugar l_0, que está representada con un punto.

Las redes de Petri permiten representar el comportamiento dinámico de un sistema. En concreto, el marcado en un instante dado representa el *estado* del sistema. Debido a las reglas de disparo, que veremos a continuación, el marcado de la red cambia con el tiempo permitiendo representar los estados por

los que va pasando el sistema.

Para formalizar la función de disparo, introducimos la noción de lugares de *entrada* y de *salida* para las transiciones. Sea (P,T,F) una red de Petri. Cada transición $t \in T$ tiene un conjunto de lugares de entrada definido por $\bullet t = \{p \in P \mid (p,t) \in F\}$ y un conjunto de lugares de salida definido por $t\bullet = \{p \in P \mid (t,p) \in F\}$. En la Figura 1.1, por ejemplo, la transición d tiene sólo un lugar de entrada, el lugar l_2, y dos lugares de salida, los lugares l_4 y l_5.

Definición 4. *(Función de disparo)* Sea (N,M) una red de Petri marcada con $N = (P,T,F)$ y $M \in \mathscr{B}(P)$. Una transición $t \in T$ está *habilitada*, expresado como $(N,M)[t\rangle$, si y solo si $\bullet t \leq M$. La *función de disparo* $\_[\_\rangle\_ \subseteq \mathscr{N} \times T \times \mathscr{N}$ es la relación más pequeña que satisface para cualquier $(N,M) \in \mathscr{N}$ y cualquier $t \in T$, $(N,M)[t\rangle \implies (N,M)[t\rangle(N,(M \setminus \bullet t) \uplus t\bullet)$.

Nota: $(N,M)[t\rangle$ denota que t está habilitada en el marcado M. $(N,M)[t\rangle(N,M')$ denota que disparar esta transición habilitada resulta en el marcado M'.

La función de disparo podemos describirla del siguiente modo. Una transición en una red de Petri se puede disparar cuando está habilitada, es decir, cuando cada uno de sus lugares de entrada tiene al menos una marca. Al dispararse, la transición consume una marca de cada uno de sus lugares de entrada y genera una marca en cada uno de sus lugares de salida, creando un nuevo marcado. Cada disparo de una transición actualiza el marcado de la red, reflejando su estado en cada momento.

El marcado permite visualizar la evolución de los estados de un sistema, indicando en qué etapa se encuentra el sistema. El marcado con el que se parte sería el *marcado inicial* de la red y representaría el estado inicial del sistema. El *marcado final* sería un marcado en el que no quedan transiciones habilitadas y por lo tanto representa un estado final del sistema.

En concreto, nos interesan las redes de Petri que representan los distintos estados por los que pasa un proceso. Para ello, las transiciones van a representar las actividades que se llevan a cabo, es decir, cada uno de los pasos bien definidos en el proceso.

Por ejemplo, tal como hemos indicado anteriormente, la red de Petri de la Figura 1.1. representa un proceso de gestión de pedidos y las transiciones son las distintas actividades de este proceso. En esta figura, el marcado inicial se representa con una marca en el lugar l_0. Esta marca inicial indica que el proceso está listo para comenzar. A medida que las transiciones se disparan, las marcas se mueven entre los lugares, actualizando el marcado y mostrando el estado actual del proceso.

En el caso de la transición a, cuando se dispara, la marca en el lugar l_0 es consumida y una nueva marca aparece en el lugar l_1, lo que indica que el proceso ha avanzado a la siguiente fase. Si hay transiciones que tienen varios lugares de salida, como d, que tiene l_4 y l_5, el disparo de esta transición hace que el marcado contenga marcas en varios lugares al mismo tiempo, ilustrando el paralelismo en el proceso.

Finalmente, en este ejemplo, si la red sólo tiene una marca en el lugar l_9, entonces no se puede disparar ninguna transición, señalando que el proceso ha concluido.

Definición 5. *(Red de Petri Etiquetada)* Una *red de Petri etiquetada* $N = (P,T,F,l)$ es una red de Petri (P,T,F) con una función de etiquetado $l \in T \nrightarrow U_A$, donde U_A es algún universo de etiquetas de actividad.

En la Figura 1.1, se muestra una red de Petri etiquetada, en la que las etiquetas son los textos que aparecen junto a los rectángulos de las transiciones. A partir de ahora nos referiremos indistintamente a las transiciones a través de su nombre o su etiqueta.

Para ilustrar con más detalle la dinámica de una red de Petri, tomemos como ejemplo la secuencia de transiciones:

$$\langle \text{recibir}, \text{comprobar}, \text{confirmar}, \text{enviar}, \text{cobrar}, \text{facturar}, \text{archivar} \rangle$$

Cuando se dispara la transición *recibir*, la marca en el lugar l_0 es consumida y una nueva marca aparece en el lugar l_1. Cuando la transición *comprobar* se dispara, se consume la marca del lugar l_1 y aparece

una nueva en el lugar l_2. En este punto, como indica la secuencia de transiciones, **se ha elegido** disparar la transición *confirmar*, la marca de l_2 es consumida y aparecen una marca en l_4 y otra en l_5. Una vez en este punto, **se deben** disparar *enviar* y *cobrar* para que las marcas de l_4 y l_5 se consuman y se generen en l_6 y l_7. Estas dos marcas son consumidas al disparar la transición *facturar*, generando una nueva marca en el lugar l_8. Por último se dispara la transición *archivar*, consumiendo la marca en l_8 para generar una en l_9, lo que indica que se ha llegado al final del proceso.

Definición 6. *(Red de Sistema)* Una *red de sistema* es una terna $SN = (N, M_{init}, M_{final})$, donde $N = (P, T, F, l)$ es una red de Petri etiquetada, $M_{init} \in \mathscr{B}(P)$ es el *marcado inicial*, y $M_{final} \in \mathscr{B}(P)$ es el *marcado final*. U_{SN} es el universo de las redes de sistema.

Por ejemplo, para la Figura 1.1, el marcado inicial M_{init} se representa colocando una marca en el lugar l_0. Este marcado inicial indica que el proceso de gestión de pedidos está listo para comenzar, con una marca en el lugar que representa el inicio del proceso (lugar de entrada) y sin marcas en los demás lugares de la red. El marcado final M_{final} se alcanza cuando una única marca llega al lugar l_9 que representa el final del proceso (lugar de salida), y todos los demás lugares están vacíos. Este estado final representa la conclusión del proceso de gestión de pedidos, ya que las actividades intermedias han sido completadas y no quedan transiciones habilitadas en la red.

1.3. Registro de eventos

El descubrimiento de procesos, tal como hemos indicado, utiliza como elemento de partida los eventos registrados durante las distintas ejecuciones de un proceso.

Antes de definir lo que es un registro de eventos, comentamos la noción de *secuencia*. Dado A un conjunto de elementos, definimos A^* como el conjunto de todas las secuencias posibles, incluidas las secuencias vacías. $\mathscr{B}(A^*)$ representa el conjunto de multiconjuntos formados por secuencias de A. Por ejemplo, si $A = \{a, b\}$, un elemento en $\mathscr{B}(A^*)$ podría ser $[\langle a, b\rangle, \langle b, a, a\rangle, \langle a, b\rangle]$, donde cada elemento es una secuencia y el multiconjunto permite múltiples apariciones de una misma secuencia.

Cada evento que se registra refiere a un *caso* particular del proceso que está siendo ejecutado y a una *actividad*. Un ejemplo de evento puede ser la confirmación (actividad) de un pedido concreto (caso). Los eventos están ordenados y pueden contener información adicional. Por ejemplo, muchas técnicas de minería de procesos utilizan información extra, como el recurso (es decir, la persona o el dispositivo) que ejecuta o inicia la actividad, la marca temporal del evento, o elementos de datos registrados con el evento (por ejemplo, el tamaño de un pedido). Ya hemos comentado previamente que en este trabajo nos vamos a centrar únicamente en el control de flujo, por esta razón, la única información que interesa considerar de los eventos es la actividad que se ejecuta. Sin embargo, hay que hacer notar que los resultados que se presentan en este trabajo, pueden extenderse fácilmente a registros de eventos que contengan información adicional.

Definición 7. *(Traza, Registro de Eventos)* Sea $A \subseteq \mathscr{A}$ un conjunto de actividades. Una *traza* $\sigma \in A^*$ es una secuencia de actividades. $L \in \mathscr{B}(A^*)$ es un *registro de eventos*, es decir, un multiconjunto de trazas.

Un registro de eventos es un *multiconjunto de trazas* dado que puede haber múltiples ejecuciones que tengan la misma traza. Por ejemplo, vamos a suponer que hemos gestionado 11 pedidos distintos y durante su ejecución se ha obtenido el siguiente registro de eventos:

$$L_{\text{pedidos}} = [\ \langle \text{recibir}, \text{comprobar}, \text{confirmar}, \text{cobrar}, \text{enviar}, \text{facturar}, \text{archivar}\rangle^5,$$
$$\langle \text{recibir}, \text{comprobar}, \text{informar errores}, \text{modificar}, \text{comprobar}, \text{confirmar}, \text{cobrar}, \text{enviar}, \text{facturar}, \text{archivar}\rangle^3,$$
$$\langle \text{recibir}, \text{comprobar}, \text{confirmar}, \text{anular}\rangle^2,$$
$$\langle \text{recibir}, \text{comprobar}, \text{informar errores}, \text{anular}, \text{archivar}\rangle^1\ ]$$

En este registro aparecen 11 casos (que corresponden a cada uno de los pedidos gestionados) que corresponden a 4 trazas distintas. Por lo tanto, L_{pedidos} representa un conjunto de eventos observados:

- $\langle$recibir, comprobar, confirmar, cobrar, enviar, facturar, archivar$\rangle^5$ indica que la secuencia *recibir* $\longrightarrow$ *comprobar* $\longrightarrow$ *confirmar* $\longrightarrow$ *cobrar* $\longrightarrow$ *enviar* $\longrightarrow$ *facturar* $\longrightarrow$ *archivar* se ha observado 5 veces.
- $\langle$recibir, comprobar, informar errores, modificar, comprobar, confirmar, cobrar, enviar, facturar, archivar$\rangle^3$ indica que la secuencia *recibir* $\longrightarrow$ *comprobar* $\longrightarrow$ *informar errores* $\longrightarrow$ *modificar* $\longrightarrow$ *comprobar* $\longrightarrow$ *confirmar* $\longrightarrow$ *cobrar* $\longrightarrow$ *enviar* $\longrightarrow$ *facturar* $\longrightarrow$ *archivar* se ha observado 3 veces.
- $\langle$recibir, comprobar, confirmar, anular$\rangle^2$ indica que la secuencia *recibir* $\longrightarrow$ *comprobar* $\longrightarrow$ *confirmar* $\longrightarrow$ *anular* se ha observado 2 veces.
- $\langle$recibir, comprobar, informar errores, anular, archivar$\rangle^1$ indica que la secuencia *recibir* $\longrightarrow$ *comprobar* $\longrightarrow$ *informar errores* $\longrightarrow$ *anular* $\longrightarrow$ *archivar* se ha observado 1 vez.

En la minería de procesos, los registros de eventos son los *elementos* fundamentales a partir de los cuales operan los algoritmos de descubrimiento. El objetivo es, a partir de un registro de eventos, determinar el modelo que corresponde con dichas ejecuciones. De este modo, el modelo obtenido recoge el comportamiento observado, por lo que describe como se está ejecutando un proceso en la realidad (modelo *de facto*). En los dos siguientes capítulos presentamos dos algoritmos de descubrimiento de procesos a partir de un registro de eventos.

Capítulo 2

Algoritmo α

El algoritmo α fue uno de los primeros algoritmos de descubrimiento de procesos que pudo abordar adecuadamente la concurrencia. Además, hay que destacar que es un algoritmo simple y que muchas de sus ideas han sido incorporadas en métodos más complejos y robustos. Por otro lado, aunque presenta problemas que comentaremos más adelante, proporciona los elementos necesarios para realizar una buena introducción al tema. Por estas razones, usaremos el algoritmo α como base para discutir los desafíos relacionados con el descubrimiento de procesos.

La idea básica del algoritmo α es escanear el registro de eventos en busca de patrones particulares. Por ejemplo, si la actividad a está seguida de la b pero la b nunca está seguida de la a, se asume que hay una dependencia causal entre a y b. Las **referencias** que hemos utilizado principalmente para desarrollar este capítulo han sido [1], [2], [4], [5], [6] y [8].

2.1. Relaciones de orden y huella de un registro

Para definir el algoritmo α, necesitamos introducir previamente las relaciones de orden para registros de eventos:

Definición 8. Sea $L \in \mathscr{B}(A^*)$ un registro de eventos sobre A y sea $a,b \in A$. Definimos las siguientes relaciones de orden basadas en L:

- $a >_L b$ si y solo si hay una traza $\sigma = \langle t_1, t_2, ...t_n \rangle$ e $i \in \{1,...,n-1\}$ tales que $\sigma \in L$, $t_i = a$ y $t_{i+1} = b$.
- $a \rightarrow_L b$ si y solo si $a >_L b$ y $b \not>_L a$.
- $a \#_L b$ si y solo si $a \not>_L b$ y $b \not>_L a$.
- $a \|_L b$ si y solo si $a >_L b$ y $b >_L a$.

Por ejemplo si consideramos el registro de eventos $L_1 = [\langle a,b,c,d \rangle^3, \langle a,c,b,d \rangle^2, \langle a,e,d \rangle]$, para este registro de eventos podemos encontrar las siguientes relaciones de orden:

$$
\begin{aligned}
>_{L_1} &= \{(a,b),(a,c),(a,e),(b,c),(c,b),(b,d),(c,d),(e,d)\} \\
\rightarrow_{L_1} &= \{(a,b),(a,c),(a,e),(b,d),(c,d),(e,d)\} \\
\#_{L_1} &= \{(a,a),(a,d),(b,b),(b,e),(c,c),(c,e),(d,a),(d,d),(e,b),(e,c),(e,e)\} \\
\|_{L_1} &= \{(b,c),(c,b)\}
\end{aligned}
$$

La relación $>_{L_1}$ contiene todos los pares de actividades en una relación de *seguimiento directo*. Por ejemplo, $c >_{L_1} d$ porque d sigue directamente a c en la traza (a,b,c,d). Sin embargo, $d \not>_{L_1} c$ porque c nunca sigue directamente a d en ninguna traza del registro. La relación $\rightarrow_{L_1}$ contiene todos los pares

de actividades en una relación de *causalidad*, por ejemplo, $c \rightarrow_{L_1} d$ porque a veces d sigue directamente a c y nunca al revés ($c >_{L_1} d$ y $d \not>_{L_1} c$). La relación $\|_{L_1}$ contiene todos los pares de actividades que *se siguen mutuamente entre sí*. Por ejemplo, $b \|_{L_1} c$ porque $b >_{L_1} c$ y $c >_{L_1} b$, es decir, a veces c sigue a b y a veces al revés. La relación $\#_{L_1}$ contiene los pares de actividades que *no se siguen de forma directa* entre ellas, $b \#_{L_1} e$ porque $b \not>_{L_1} e$ y $e \not>_{L_1} b$.

Para cualquier registro L sobre A y $x, y \in A$: $x \rightarrow_L y$, $y \rightarrow_L x$, $x \#_L y$, o $x \|_L y$, es decir, para cada par de actividades se cumple exactamente una de estas relaciones. Por lo tanto, las relaciones entre las actividades de un registro se pueden capturar mediante una matriz a la que llamaremos *huella*. Por ejemplo, la huella del registro L_1 es la siguiente:

	a	**b**	**c**	**d**	**e**
a	#	$\rightarrow$	$\rightarrow$	#	$\rightarrow$
b	$\leftarrow$	#	$\|\|$	$\rightarrow$	#
c	$\leftarrow$	$\|\|$	#	$\rightarrow$	#
d	#	$\leftarrow$	$\leftarrow$	#	$\leftarrow$
e	$\leftarrow$	#	$\leftarrow$	$\rightarrow$	#

Tabla 2.1: Huella del registro L_1.

Las relaciones de orden que acabamos de definir se pueden relacionar con diversos patrones de las redes de Petri. En concreto, son los siguientes:

- *Patrón secuencial*: Si en la red de Petri a y b están en secuencia, entonces en la huella del registro aparecerá la relación $a \rightarrow_L b$ (Figura 2.1).
- *Patrones XOR-división y XOR-unión*: Por un lado, si en la red de Petri después de a hay una elección entre b y c (patrón XOR-división), el registro mostrará $a \rightarrow_L b$, $a \rightarrow_L c$, y $b \nparallel_L c$ porque a puede ser seguido por b y por c, pero b no será seguido por c y viceversa (Figura 2.2). Por el contrario, si después de la ocurrencia de a o b debe ocurrir c (patrón XOR-unión), entonces en la huella aparecerá $a \rightarrow_L c$, $b \rightarrow_L c$, y $a \nparallel_L b$, (Figura 2.3).
- *Patrones AND-división y AND-unión*: Por una parte, si en la red de Petri después de a, tanto b como c pueden ser ejecutados en paralelo (patrón AND-división), el registro mostrará $a \rightarrow_L b$, $a \rightarrow_L c$, y $b \|_L c$ (Figura 2.4). Por otra parte, si en la red se muestra la unión de a y b para ejecutar c (patrón AND-unión), el registro mostrará $a \rightarrow_L c$, $b \rightarrow_L c$, y $a \|_L b$ (Figura 2.5).

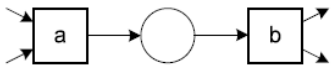

Figura 2.1: Patrón secuencial.

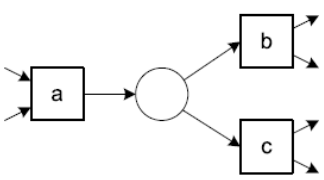

Figura 2.2: Patrón XOR-división.

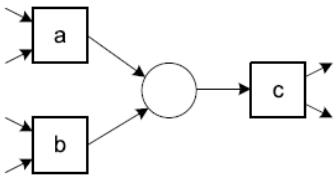

Figura 2.3: Patrón XOR-unión.

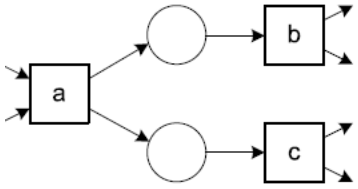

Figura 2.4: Patrón AND-división.

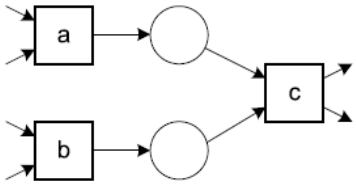

Figura 2.5: Patrón AND-unión.

Una vez vistas estas nociones vamos a presentar a continuación el algoritmo alpha.

2.2. Definición del algoritmo α

Definición 9. (*Algoritmo* α) Sea $L \in \mathscr{B}(A^*)$ un registro de eventos sobre A. $\alpha(L)$ produce una red de sistema y se define como sigue:

1. $T_L = \{t \in A \mid \exists_{\sigma \in L} \text{ tal que } t \in \sigma\}$,
2. $T_I = \{t \in A \mid \exists_{\sigma \in L} \text{ tal que } t = \text{primero}(\sigma)\}$,
3. $T_O = \{t \in A \mid \exists_{\sigma \in L} \text{ tal que } t = \text{último}(\sigma)\}$,
4. $X_L = \{(A,B) \mid A \subseteq T_L \wedge A \neq \emptyset \wedge B \subseteq T_L \wedge B \neq \emptyset \wedge \forall a \in A \forall b \in B\ a \rightarrow_L b \wedge \forall a_1, a_2 \in A\ a_1 \#_L a_2 \wedge \forall b_1, b_2 \in B\ b_1 \#_L b_2\}$,
5. $Y_L = \{(A,B) \in X_L \mid \forall (A',B') \in X_L\ A \subseteq A' \wedge B \subseteq B' \Rightarrow (A,B) = (A',B')\}$,
6. $P_L = \{p_{(A,B)} \mid (A,B) \in Y_L\} \cup \{i_L, o_L\}$,
7. $F_L = \{(a, p_{(A,B)}) \mid (A,B) \in Y_L \wedge a \in A\} \cup \{(p_{(A,B)}, b) \mid (A,B) \in Y_L \wedge b \in B\} \cup \{(i_L, t) \mid t \in T_I\} \cup \{(t, o_L) \mid t \in T_O\}$,
8. $l_L \in T_L \rightarrow A$ con $l(t) = t$ para todo $t \in T_L$,
9. $\alpha(L) = (N, M_{\text{inicial}}, M_{\text{final}})$ con $N = (P_L, T_L, F_L, l_L), M_{\text{inicial}} = [i_L], M_{\text{final}} = [o_L]$.

En base a esta definición, vamos a ir explicando el algoritmo α paso por paso para entender correctamente su funcionamiento.

- Paso 1: se determinan qué actividades aparecen en el registro (T_L). Estas son las actividades observadas y corresponden a las transiciones de la red de sistema generada. Hacer notar que, salvo en el paso 8, usaremos indistintamente el término actividad y transición. En el caso del registro L_1, T_{L_1} es el conjunto de actividades, es decir, $T_{L_1} = \{a, b, c, d, e\}$
- Paso 2: determina qué actividades aparecen primero en alguna traza, T_I es el conjunto de actividades de inicio. En el caso del registro L_1, $T_{I_{L_1}} = \{a\}$.
- Paso 3: determina qué actividades aparecen al final en alguna traza, T_O es el conjunto de actividades finales. Para el registro L_1, $T_{O_{L_1}} = \{d\}$.
- Paso 4: buscamos pares de conjuntos de actividades A y B, no vacíos y cumpliendo las condiciones establecidas en la definición. Primero, para cualquier actividad del conjunto A y para cualquier actividad del conjunto B, existe una relación directa entre ellas ($a \rightarrow_L b$). Además, tomando dos actividades del conjunto A (a_1, a_2), no deben seguirse una a la otra ($a_1 \# a_2$). Por último, tomando dos actividades del conjnuto B (b_1, b_2), no deben segirse una a la otra ($b_1 \# b_2$). En la figura 2.6 aparece un ejemplo de un par de conjuntos A y B. Todos los elementos de A deben tener dependencias de causalidad con todos los elementos de B, es decir, para todo $(a,b) \in A \text{x} B : a \longrightarrow_L b$.

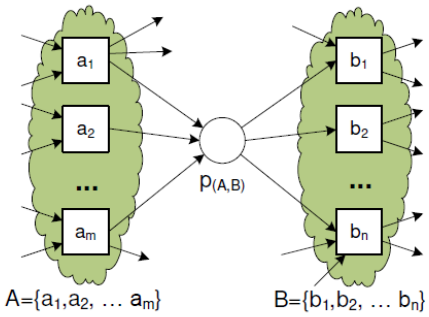

Figura 2.6: Esquema gráfico del paso 4 del algoritmo α (extraído de [1]).

Con estos dos conjuntos A y B obtenemos el patrón mostrado en la tabla 2.2, que está formado por cuatro cuadrantes. Dos cuadrantes solo contienen el símbolo #; esto indica que los elementos de A nunca deben seguir a otro (cuadrante superior izquierdo) y que los elementos de B nunca deben seguir a otro (cuadrante inferior derecho). El cuadrante superior derecho solo contiene el símbolo $\rightarrow$; cualquiera de los elementos en A puede ser seguido por cualquiera de los elementos en B, pero nunca al revés. Por simetría, el cuadrante inferior izquierdo solo contiene el símbolo $\leftarrow$.

	a_1	a_2	$\cdots$	a_m	b_1	b_2	$\cdots$	b_n
a_1	#	#	$\cdots$	#	$\rightarrow$	$\rightarrow$	$\cdots$	$\rightarrow$
a_2	#	#	$\cdots$	#	$\rightarrow$	$\rightarrow$	$\cdots$	$\rightarrow$
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
a_m	#	#	$\cdots$	#	$\rightarrow$	$\rightarrow$	$\cdots$	$\rightarrow$
b_1	$\leftarrow$	$\leftarrow$	$\cdots$	$\leftarrow$	#	#	$\cdots$	#
b_2	$\leftarrow$	$\leftarrow$	$\cdots$	$\leftarrow$	#	#	$\cdots$	#
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
b_n	$\leftarrow$	$\leftarrow$	$\cdots$	$\leftarrow$	#	#	$\cdots$	#

Tabla 2.2: Huella correspondiente a los conjuntos A y B.

En el caso del registro L_1,
$X_{L_1} = \{(\{a\},\{b\}),(\{a\},\{c\}),(\{a\},\{e\}),(\{a\},\{b,e\}),(\{a\},\{c,e\}),(\{b\},\{d\}),(\{c\},\{d\}),(\{e\},\{d\}),(\{b,e\},\{d\}),(\{c,e\},\{d\})\}$.
Por ejemplo, si nos fijamos únicamente en a, b y e, tenemos la siguiente huella:

	a	b	e
a	$\#_{L_1}$	$\rightarrow_{L_1}$	$\rightarrow_{L_1}$
b	$\leftarrow_{L_1}$	$\#_{L_1}$	$\#_{L_1}$
e	$\leftarrow_{L_1}$	$\#_{L_1}$	$\#_{L_1}$

Tabla 2.3: Huella correspondiente a los conjuntos $\{a\}$ y $\{b,e\}$.

- Paso 5: eliminamos todos los pares (A,B) que no son máximos. En el caso del registro L_1, $Y_{L_1} = \{(\{a\},\{b,e\}),(\{a\},\{c,e\}),(\{b,e\},\{d\}),(\{c,e\},\{d\})\}$
- Paso 6: se determinan los lugares de la red de sistema en base a los conjuntos definidos en el paso anterior. En concreto, se define un lugar para cada par de conjuntos (A,B), que servirá para conectar las transiciones de A con las transiciones de B. Y además se añaden dos lugares, uno

inicial y uno final. La Figura 3.2 muestra los lugares que comentamos, como lugar inicial tenemos i_L y como final o_L. En el centro de la figura encontramos el lugar $p_{(A,B)}$.
En el caso del registro L_1, $P_{L_1} = \{p_{(\{a\},\{b,e\})}, p_{(\{a\},\{c,e\})}, p_{(\{b,e\},\{d\})}, p_{(\{c,e\},\{d\})}, i_{L_1}, o_{L_1}\}$.

- Paso 7: se generan los arcos de la red de sistema, para ello conectamos el lugar de inicio i_L con las transiciones de inicio T_I y las transiciones de fin T_O con el lugar de fin o_L. Todos los lugares $p_{(A,B)}$ tienen las transiciones de A como nodos de entrada y las transiciones de B como nodos de salida. Con ello obtenemos el esquema de red de sistema mostrado en la Figura 3.2.
Para nuestro registro L_1:

$$F_{L_1} = \{(a, p_{(\{a\},\{b,e\})}), (p_{(\{a\},\{b,e\})}, b)(p_{(\{a\},\{b,e\})}, e), (a, p_{(\{a\},\{c,e\})}), (p_{(\{a\},\{c,e\})}, c),$$
$$(p_{(\{a\},\{c,e\})}, e), (b, p_{(\{b,e\},\{d\})}), (e, p_{(\{b,e\},\{d\})}), (p_{(\{b,e\},\{d\})}, d), (c, p_{(\{c,e\},\{d\})}),$$
$$(e, p_{(\{c,e\},\{d\})}), (p_{(\{c,e\},\{d\})}, d), (i_{L_1}, a), (d, o_{L_1})\}$$

- Paso 8: como la red de sistema es una red de Petri etiquetada, se define la función de etiquetado (l_L) de la red simplemente asignando a cada transición la actividad que representa.
En el caso de L_1, las transiciones tendrán como etiquetas a, b, c, d y e.

- Paso 9: en el paso final se determina la red de sistema resultante, formada por los lugares definidos en P_L, las transiciones de T_L, los arcos F_L y la función de etiquetado l_L, junto con el marcado inicial y final. Para el registro L_1 obtenemos:

$$\alpha(L_1) = (N, [i_{L_1}], [o_{L_1}]), \text{ con } N = (P_{L_1}, T_{L_1}, F_{L_1}, l_{L_1}) \tag{2.1}$$

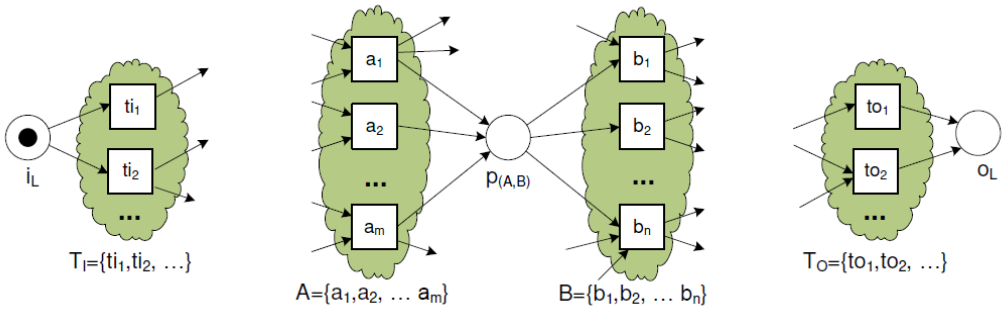

Figura 2.7: Esquema gráfico de algunos elementos utilizados en el algoritmo α (extraído de [2]).

En la figura 2.8 aparece la representación gráfica de la red de sistema que resulta al aplicar el algoritmo α al registro L_1, indicando el marcado inicial de la red. En concreto, esta imagen se ha obtenido usando la herramienta ProM 6.13[1]. Se puede observar que realmente esta red de sistema puede reproducir todas las trazas del registro de eventos de L_1. De hecho, las tres únicas posibles secuencias de disparo de la red son las que aparecen en L_1.

[1] `https://www.promtools.org`

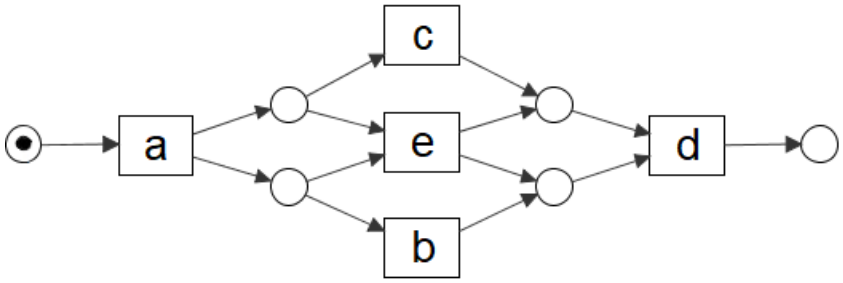

Figura 2.8: Red de Petri resultante de L_1.

2.3. Limitaciones del algoritmo α en Minería de Procesos

Una primera limitación del algoritmo α es que no puede descubrir lo que no se ha registrado. Por lo tanto, es conveniente que los registros sean completos, es decir, que contengan todas las trazas posibles, para que puedan obtenerse modelos que representen adecuadamente el comportamiento del sistema. Además, el algoritmo α tiene varios problemas adicionales al intentar extraer modelos precisos de los registros de eventos. Estas limitaciones hacen que el algoritmo no siempre sea capaz de generar una red de Petri que represente de manera exacta lo que realmente aparece en el log de eventos, especialmente en el caso de ciertas estructuras complejas de la red del sistema. A continuación, se explican algunas de las principales dificultades:

- **Bucles cortos**: El algoritmo α tiene problemas para trabajar con bucles de longitud uno o dos. En bucles de un solo paso, una actividad se repite de forma consecutiva, pero el algoritmo no puede crear una estructura de este tipo porque necesitaría que una transición a tuviera una relación $a \rightarrow a$ que es imposible (tendría que ocurrir $a > a$ y $a \not> a$ al mismo tiempo). En bucles de dos pasos, infiere que las tareas están en paralelo, ignorando las relaciones de causalidad necesarias.
- **Actividad silenciosa**: Las actividades silenciosas funcionan como puntos de enrutamiento en el flujo pero no están registradas explícitamente en el registro, por lo que no pueden ser detectadas por el algoritmo α. Esto deja el modelo incompleto, ya que estas actividades son importantes para la estructura del flujo, tal como veremos en el siguiente capítulo.
- **Lugares implícitos**: Los lugares implícitos son aquellos que su presencia o ausencia no afecta a las posibles trazas que se pueden generar con el modelo. Como el algoritmo α crea lugares basándose en relaciones de causalidad, estos lugares no se detectan, limitando la precisión del modelo.
- **Elección no libre**: Este tipo de estructura combina sincronización y elección, lo que hace que el algoritmo α no pueda representar bien situaciones donde una decisión depende de varios factores sin una causalidad clara. Esto hace que el algoritmo α no siempre genere adecuadamente las redes que contienen elecciones no libres.

Estas limitaciones ponen de manifiesto la necesidad de enfoques alternativos o adaptaciones del algoritmo α para poder representar mejor una mayor variedad de estructuras en las redes de sistema.

A modo de ejemplo, vamos a presentar a continuación dos registros de eventos para los que el algoritmo α genera una red de Petri inadecuada. El primer ejemplo corresponde a un registro de eventos con un bucle de longitud uno. El registro es el siguiente:

$$L_2 = \left[\langle a,c\rangle^2, \langle a,b,c\rangle^3, \langle a,b,b,c\rangle^2, \langle a,b,b,b,b,c\rangle^1\right]$$

En este registro se observa que la actividad b se puede ejecutar repetidas veces entre la actividad a y la actividad c. Sin embargo la red que genera el algoritmo α aparece en la figura 2.9, en la que se puede observar que la actividad b está desconectada de la red, de modo que permite la ejecución de b antes de a y después de c. Este comportamiento que permite la red no es consistente con el registro de eventos.

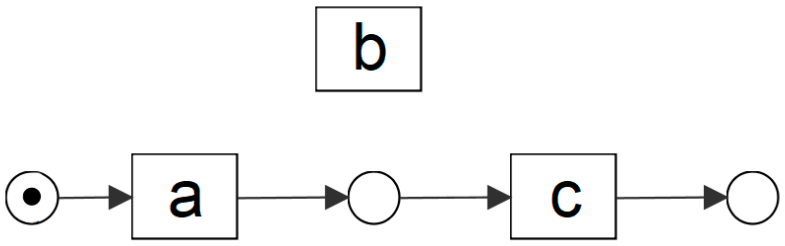

Figura 2.9: Red de Petri resultante de aplicar algoritmo α a L_2.

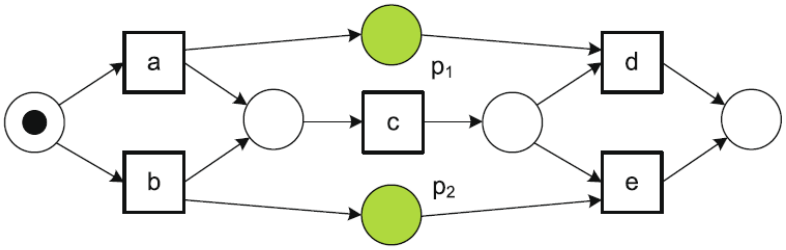

Figura 2.10: Red de Petri con una elección no libre (extraído de [1]).

El segundo ejemplo muestra una red con una elección no libre que no es generada adecuadamente por el algoritmo α. En concreto, suponiendo que partimos de la red de Petri de la figura 2.10, un posible registro de eventos sería el siguiente:

$$L_3 = \left[\langle a,c,d\rangle^{45}, \langle b,c,e\rangle^{42}\right]$$

En esta red, para que aparezca la actividad d en el registro, es necesario que antes haya ocurrido a. Lo mismo pasa con la actividad e y b. Sin embargo, al aplicar el algoritmo α, la red resultante es la de la figura 2.11, en la que no aparecen los lugares p_1 y p_2 de la figura 2.10, de modo que permite trazas que no debería, por ejemplo $\langle a,c,e\rangle$ y $\langle b,c,d\rangle$.

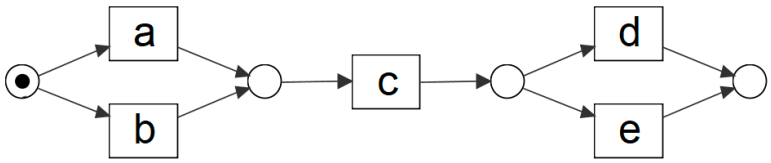

Figura 2.11: Red de Petri resultante de aplicar algoritmo α a L_3.

Capítulo 3

Descubrimiento de Procesos Inductivo

El descubrimiento de procesos inductivo es una técnica que utiliza una estrategia de *divide y vencerás*. Esto significa que, en lugar de analizar todo el registro de eventos de una sola vez, lo divide en bloques más pequeños para procesarlos de forma independiente. A través de estos bloques, se van descubriendo las relaciones entre las actividades, como por ejemplo el orden en que se realizan, cuáles son opcionales o cuáles pueden hacerse en cualquier orden. Como resultado se obtiene un árbol de procesos que es un modelo estructurado en bloques. Los árboles de procesos, a diferencia de otras técnicas de modelado de procesos, garantiza que los modelos creados sean correctos desde el principio, lo que significa que no tendrán problemas como bloqueos o errores en las conexiones entre actividades. Una vez obtenido el árbol de procesos, éste puede ser fácilmente transformado en otros modelos, como por ejemplo en una red de Petri.

Una de las ventajas del descubrimiento inductivo es que es muy flexible y puede adaptarse a diferentes tipos de registros, incluso aquellos que son muy grandes o contienen datos poco frecuentes. Otra ventaja es que funciona bien en situaciones en las que otros métodos, como el algoritmo α, tienen dificultades. Por ejemplo, mientras que el algoritmo α necesita registros completos para funcionar correctamente, el descubrimiento de procesos inductivo puede adaptarse a datos incompletos o con errores. Esto lo hace especialmente útil para analizar procesos reales, donde los datos no siempre son perfectos.

La **referencia** que hemos utilizado principalmente para desarrollar este capítulo ha sido [1].

3.1. Algoritmo de Minería Inductiva

Para explicar el algorirmo de minería inductiva, vamos a tomar un ejemplo de registro para construir una red paso por paso. Como ejemplo hemos tomado el registro:

$$L_{\text{IM}} = \{(a,b,c,d)^3, (a,c,b,d)^4, (a,b,c,e,f,b,c,d)^2, (a,c,b,e,f,b,c,d)^2, (a,c,b,e,f,b,c,e,f,c,b,d), (a,g,d)\}$$

El algoritmo básico de minería inductiva utiliza el grafo de seguimiento directo que corresponde a la relación de "seguimiento directo" ($>_L$) usada por el algoritmo α (ver definición 9). Por esta razón, comenzamos analizando elementos similares a los empleados en el algoritmo α, como la relación de seguimiento directo, así como las actividades iniciales y finales.

Definición 10. (*Grafo de seguimiento directo*) Sea L un registro de eventos, es decir, $L \in \mathscr{B}(A^*)$. El *grafo de seguimiento directo* de L es $G(L) = (A_L, \mapsto_L, A_L^{\text{inicio}}, A_L^{\text{fin}})$, donde:

- $A_L = \{a \in \sigma \mid \sigma \in L\}$ es el conjunto de actividades en L,
- $\mapsto_L = \{(a,b) \in A \times A \mid a >_L b\}$ es la relación de seguimiento directo,

- $A_L^{\text{inicio}} = \{a \in A \mid \exists \sigma \in L : a = \text{primero}(\sigma)\}$ es el conjunto de actividades de inicio, y
- $A_L^{\text{fin}} = \{a \in A \mid \exists \sigma \in L : a = \text{último}(\sigma)\}$ es el conjunto de actividades de fin.

Además, también vamos a considerar la relación $\mapsto^+$ que representa el cierre transitivo de $\mapsto$. Por ejemplo, $a \mapsto^+ d$ porque la primera traza de L_{IM} describe un camino desde a hasta d.

Antes de mostrar el grafo de seguimiento directo correspondiente a L_{IM}, vamos a definir la noción de *corte*, así podremos mostrar en conjunto el gráfico de seguimiento directo con sus respectivos cortes y poder explicar todo el proceso detalladamente.

Definición 11. (*Corte*) Sea L un registro de eventos con su grafo de seguimiento directo correspondiente $G(L) = (A_L, \mapsto_L, A_L^{\text{inicio}}, A_L^{\text{fin}})$. Sea $n \geq 1$. Un *n-corte* de $G(L)$ es una partición de A_L en subconjuntos disjuntos $A_1, A_2, \ldots, A_n$, tal que $A_L = \bigcup_{i=1}^{n} A_i$ y $A_i \cap A_j = \emptyset$ para $i \neq j$. La notación es $(\oplus, A_1, A_2, \ldots, A_n)$ con $\oplus \in \{\rightarrow, \times, \wedge, \circlearrowleft\}$. Para cada tipo de operador $(\rightarrow, \times, \wedge, \circlearrowleft)$ se aplican condiciones específicas:

- Un *corte de elección exclusiva* de $G(L)$ es un corte $(\times, A_1, A_2, \ldots, A_n)$ tal que:
  - $\forall i, j \in \{1, \ldots, n\}, \forall a \in A_i, \forall b \in A_j, i \neq j \Rightarrow a \not\mapsto_L b$.
- Un *corte secuencial* de $G(L)$ es un corte $(\rightarrow, A_1, A_2, \ldots, A_n)$ tal que:
  - $\forall i, j \in \{1, \ldots, n\}, \forall a \in A_i, \forall b \in A_j, i < j \Rightarrow (a \mapsto_L^+ b \wedge b \not\mapsto_L^+ a)$.
- Un *corte paralelo* de $G(L)$ es un corte $(\wedge, A_1, A_2, \ldots, A_n)$ tal que:
  - $\forall i \in \{1, \ldots, n\}, A_i \cap A_L^{\text{inicio}} \neq \emptyset \wedge A_i \cap A_L^{\text{fin}} \neq \emptyset$, y
  - $\forall i, j \in \{1, \ldots, n\}, \forall a \in A_i, \forall b \in A_j, i \neq j \Rightarrow a \mapsto_L b$.
- Un *corte de bucle de repetición* de $G(L)$ es un corte $(\circlearrowleft, A_1, A_2, \ldots, A_n)$ tal que:
  - $n \geq 2$,
  - $A_L^{\text{inicio}} \cup A_L^{\text{fin}} \subseteq A_1$,
  - $\{a \in A_1 \mid \exists i \in \{2, \ldots, n\}, \exists b \in A_i : a \mapsto_L b\} \subseteq A_L^{\text{fin}}$,
  - $\{a \in A_1 \mid \exists i \in \{2, \ldots, n\}, \exists b \in A_i : b \mapsto_L a\} \subseteq A_L^{\text{inicio}}$,
  - $\forall i, j \in \{2, \ldots, n\}, \forall a \in A_i, \forall b \in A_j, i \neq j \Rightarrow a \not\mapsto_L b$,
  - $\forall i \in \{2, \ldots, n\} \forall b \in A_i \exists a \in A_L^{\text{fin}}\, a \mapsto_L b \implies \forall a' \in A_L^{\text{fin}}\, a' \mapsto_L b$, y
  - $\forall i \in \{2, \ldots, n\} \forall b \in A_i \exists a \in A_L^{\text{inicio}}\, b \mapsto_L a \implies \forall a' \in A_L^{\text{inicio}}\, b \mapsto_L a'$.

Un corte $(\oplus, A_1, A_2, \ldots, A_n)$ con $\oplus \in \{\rightarrow, \times, \wedge, \circlearrowleft\}$ del grafo de seguimiento directo $G(L)$ es *máximo* si no existe ningún corte $(\oplus, A_1', A_2', \ldots, A_m')$ con $m > n$. El corte $(\oplus, A_1, A_2, \ldots, A_n)$ es llamado *trivial* si $n = 1$.

De acuerdo a estas definiciones, para nuestro registro L_{IM}, obtenemos el siguiente grafo de seguimiento directo, $G(L_{IM})$, donde los cortes vienen representados por líneas discontinuas rojas.

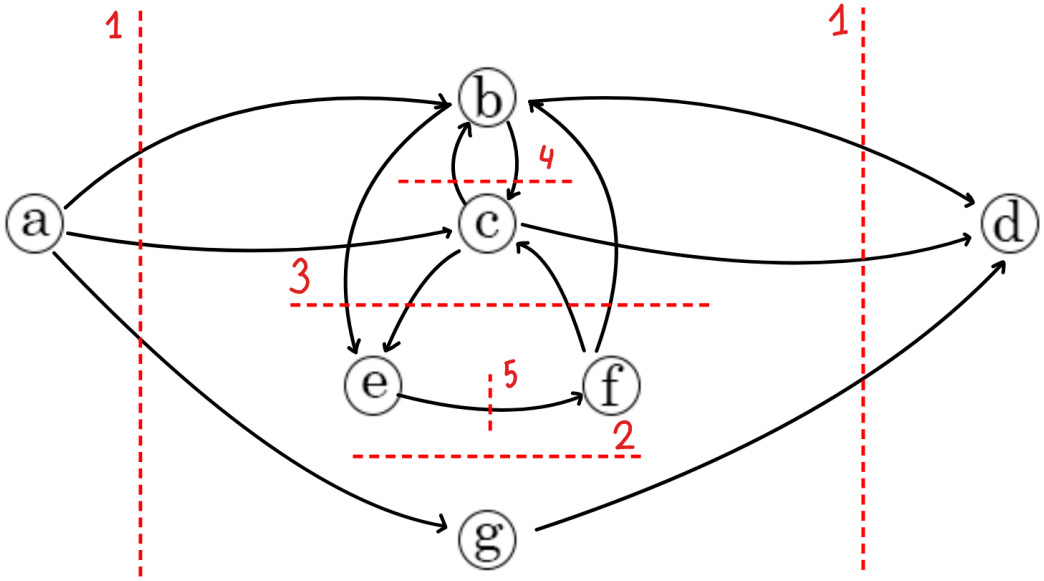

Figura 3.1: Grafo de seguimiento directo correspondiente a L_{IM}.

A continuación vamos a analizar los cortes uno por uno para comprender finalmente cómo se realizan.

1. Se corresponde con el *corte secuencial*, porque tenemos un camino desde a hasta cada uno de los elementos de la parte central, y desde los elementos centrales hasta d, pero no existen caminos en el sentido contrario, representando que tenemos una única dirección. Por lo tanto cortamos justo a la derecha de a y a la izquierda de d, separando el grafo en tres partes.

2. Nos quedamos con la parte central de la figura, excluyendo a y d, que ya los hemos cortado antes. Este segundo corte representa un *corte de elección exclusiva*. No existe ningún arco entre los elementos de la parte superior y g, por lo que tenemos la posibilidad de ir al bloque superior (formado por b, c, e y f) o de ir al bloque inferior, en el que tenemos solo g.

3. Ahora nos centramos sólo en b, c, e, f. En este paso se lleva a cabo un *corte de bucle de repetición*, vamos a repasar las condiciones para verificar que efectivamente se cumplen. En primer lugar notar que tenemos entrada desde a y salida a d únicamente para el primer bloque (formado por b y c), dejando el segundo bloque con e y f. Si sale una arista del primer bloque a otro bloque, entonces tiene que ser entrada del subgrafo completo, y efectivamente, de b y c salen aristas al segundo bloque y además les llega una flecha desde a. Lo mismo ocurre si les llega una flecha del segundo bloque. A b y c les llegan flechas de e y f y por lo tanto tienen que tener flechas de salida a d (tal como ocurre). En el caso de que hubiese más de dos bloques, sería necesario que no existieran flechas entre los bloques (del bloque 2 en adelante).

4. Se corresponde con un *corte paralelo*, tenemos entrada y salida en cada bloque. Del b se va a c y viceversa.

5. Igual que en el primer corte, este es un *corte secuencial*, de e a f en una única dirección, es decir, sin camino de vuelta.

Antes de pasar a introducir el concepto de árbol de procesos necesitamos definir el concepto de *actividad silenciosa*. Una actividad silenciosa, denotada por τ, es una actividad que no puede ser observada. Por ejemplo, un árbol de procesos $\times(a, \tau)$ puede usarse para modelar una actividad a que puede ser omitida. De manera similar, la repetición $\circlearrowleft(a, \tau)$ modela un proceso que ejecuta a cualquier número de veces y también es posible no ejecutar a en absoluto. En las redes de Petri, las actividades silenciosas se representan con rectángulos negros o con la letra τ dentro de la caja.

Definición 12. (*Árbol de proceso*) Sea $A \subseteq \mathscr{A}$ un conjunto finito de actividades con $\tau \notin A$. Consideramos $\oplus = \{\rightarrow, \times, \wedge, \circlearrowleft\}$ como el conjunto de *operadores de árboles de proceso*.

- Si $a \in A \cup \{\tau\}$, entonces $Q = a$ es un árbol de proceso.
- Si $n \geq 1$, $Q_1, Q_2, \ldots, Q_n$ son árboles de proceso, y $\oplus \in \{\rightarrow, \times, \wedge\}$, entonces $Q = \oplus(Q_1, Q_2, \ldots, Q_n)$ es un árbol de proceso.
- Si $n \geq 2$ y $Q_1, Q_2, \ldots, Q_n$ son árboles de proceso, entonces $Q = \circlearrowleft (Q_1, Q_2, \ldots, Q_n)$ es un árbol de proceso.

$\mathscr{Q}_A$ es el conjunto de todos los *árboles de proceso* sobre A.

Existen cuatro tipos de operadores en los árboles de procesos que permiten estructurar el flujo de actividades, que se corresponden con los tipos de corte que hemos visto para los grafos de seguimiento directo. El operador de *secuencia*, representado con la flecha hacia la derecha ($\rightarrow$), establece un orden estricto entre actividades, donde una debe seguir a la otra. El operador de *paralelismo* se representa con una $\wedge$, permitiendo que dos o más actividades puedan ejecutarse simultáneamente o en cualquier orden. El operador de *selección* (o exclusión), simbolizado con una $\times$, significa que se debe escoger una entre varias actividades posibles, excluyendo las otras. Finalmente, el operador de *bucle*, representado como $\circlearrowleft$, facilita la repetición de una actividad o conjunto de actividades. Estos operadores permiten modelar distintos comportamientos en los procesos, adaptándose a sus características y restricciones.

Por lo tanto, y teniendo en cuenta los cortes de la figura 3.1, el árbol de procesos del registro L_{IM} que obtenemos es::

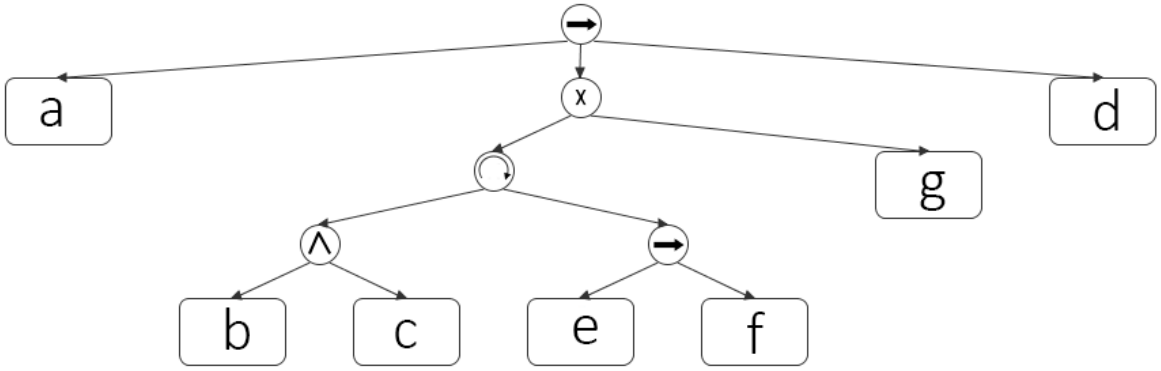

Figura 3.2: Árbol de procesos correspondiente a L_{IM}.

Una vez que tenemos el árbol de procesos, ya estamos en condiciones de generar la red de Petri correspondiente a L_{IM}. Los árboles de procesos pueden ser convertidos en redes de Petri mediante las transformaciones que se muestran en la siguiente figura:

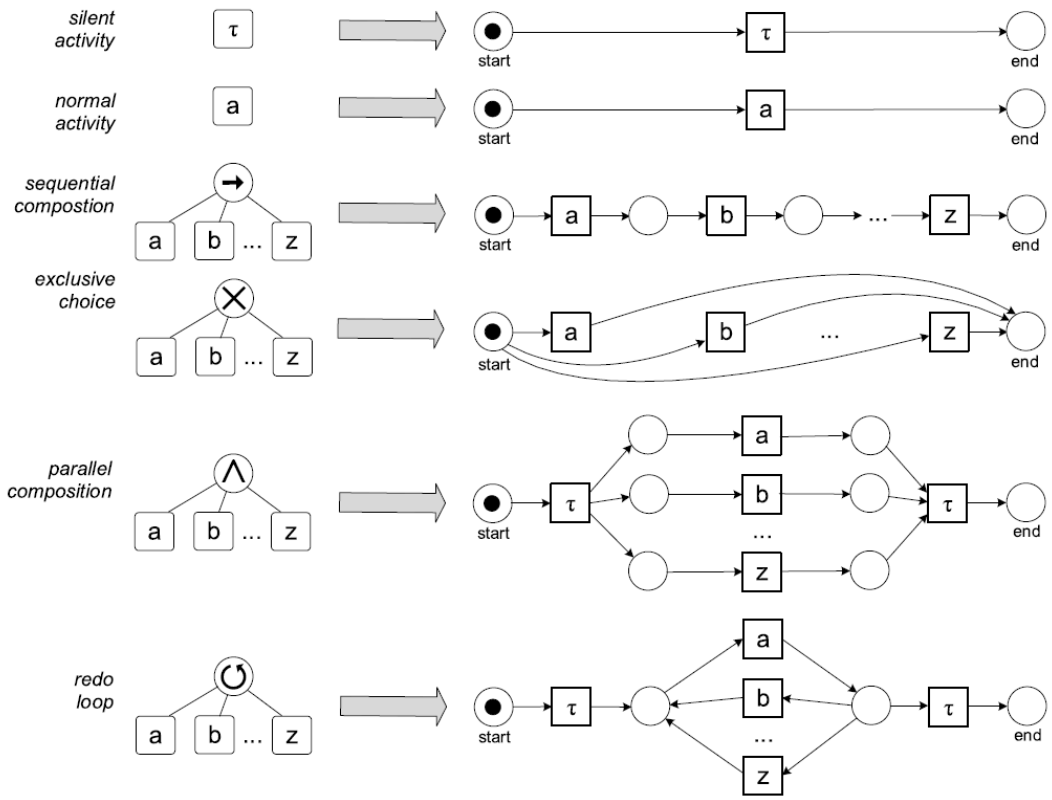

Figura 3.3: Transformación de un árbol de procesos en una red de Petri (extraído de [1]).

Haciendo uso de estas reglas de transformación, a partir del árbol de procesos de L_{IM} (figura 3.3) se obtiene la red de Petri de la figura 3.4. En concreto, se puede construir la red, empezando por las hojas centrales del árbol de procesos, transformando el paralelismo $\wedge(b,c)$ y la secuencia $\rightarrow(e,f)$. Seguido de ello continuamos hacia arriba en el árbol y transformamos el bucle $\circlearrowleft(\wedge(b,c),\rightarrow(e,f))$. Finamente transformamos la selección $\times(\circlearrowleft(\wedge(b,c),\rightarrow(e,f)),g)$ y la secuencia $\rightarrow(a,\times(\circlearrowleft(\wedge(b,c),\rightarrow(e,f)),g),d)$.

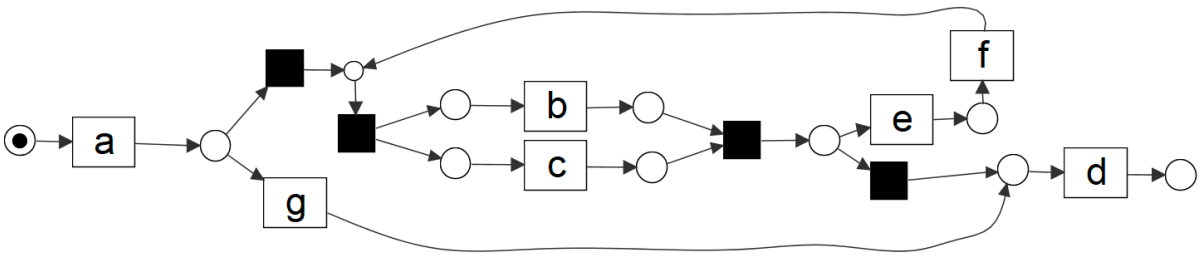

Figura 3.4: Red de Petri correspondiente a L_{IM}.

3.2. Límites del algoritmo de minería inductiva

El algoritmo de minería inductiva resuelve bastantes de los problemas del algoritmo α, pero no todos. En concreto, a modo de ejemplo, vamos a utilizar este algoritmo para los registros L_2 y L_3 que vimos en el apartado 2.3.

En la figura 3.5 aparece la red de Petri que se obtiene al aplicar el algoritmo de minería inductiva al registro L_2. En la imagen se observa que la red de Petri sí representa adecuadamente el bucle de longitud 1. Precisamente la forma de representarlo ha consistido en utilizar una actividad silenciosa.

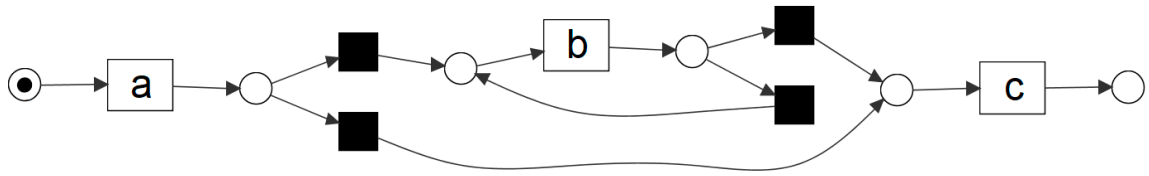

Figura 3.5: Red de Petri resultante de aplicar algoritmo inductivo a L_2.

En la figura 3.6 aparece la red de Petri que se obtiene al aplicar el algoritmo de minería inductiva al registro L_3. En la imagen se observa que la red de Petri sigue sin representar adecuadamente la elección no libre, puesto que no ha descubierto los lugares p_1 y p_2 de la red de Petri de la figura 2.10. Se observa que es la misma figura que obteníamos con el algoritmo α.

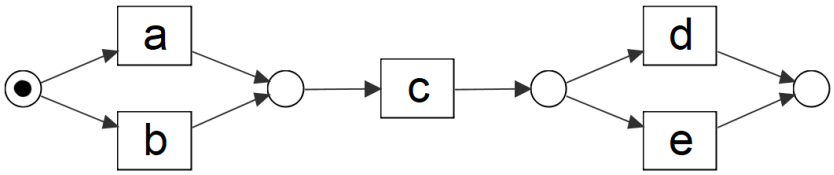

Figura 3.6: Red de Petri resultante de aplicar algoritmo inductivo a L_3.

Capítulo 4

Aplicación práctica de los algoritmos

Para este capítulo vamos a utilizar la aplicación de minería de procesos ProM 6.13[1] y el registro de eventos (obtenido de una aplicación real) Road Traffic Fine Management Process[2], que está disponible públicamente. En concreto, vamos a aplicar un plugin de ProM que implementa el algoritmo α, y un plugin que implementa un algoritmo de minería inductiva. El objetivo es comparar los resultados obtenidos al usar estos dos plugins sobre un mismo registro de eventos reales.

El registro de eventos que vamos a utilizar contiene información sobre el proceso de gestión de multas de tráfico. Cada registro representa un caso único, correspondiente a una multa, y está compuesto por una serie de eventos que describen las distintas etapas del proceso, como la emisión de la multa, la notificación al infractor, el pago y posibles acciones legales. Cada evento incluye detalles como un identificador único del caso, un sello de tiempo que indica el momento del evento y la actividad realizada. Este registro de eventos proporciona un historial detallado de las actividades relacionadas con la gestión de multas. Los datos provienen de un sistema real de gestión de multas de tráfico implementado en el gobierno local de Italia. Este sistema está diseñado para gestionar todo el ciclo de vida de una multa, desde su emisión hasta su resolución.

Este registro de eventos real contiene 561.470 eventos agrupados en 3.734 trazas distintas.

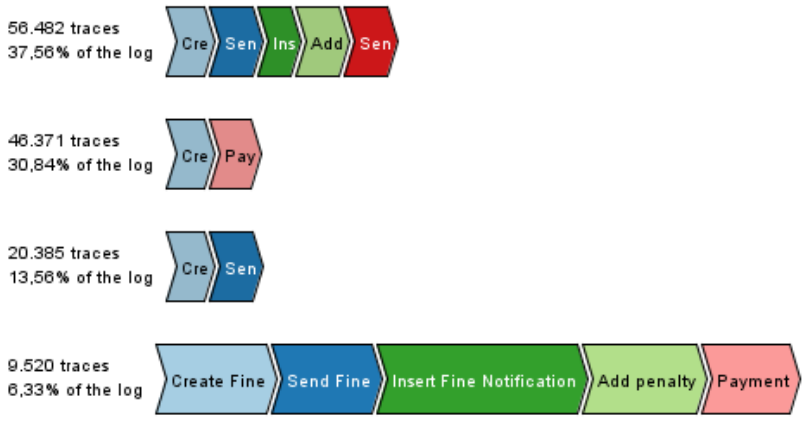

Figura 4.1: Análisis de las trazas del registro de eventos.

En la figura 4.1 se muestran algunas de las trazas que forman parte del registro de eventos, que representan el 88,29 % de los casos. Por ejemplo, si nos fijamos en la última traza de la imagen, se observa que ha ocurrido 9.520 veces, lo cual corresponde al 6.33 % del registro de eventos. En concreto, esta traza está formada por los eventos *crear multa, enviar multa, insertar notificación de multa, agregar penalización y pago*.

[1] `https://www.promtools.org`
[2] `https://data.4tu.nl/articles/_/12683249/1`

Utilizando la herramienta de ProM de nuevo y aplicando el algoritmo α a este registro obtenemos la red de Petri de la figura 4.2.

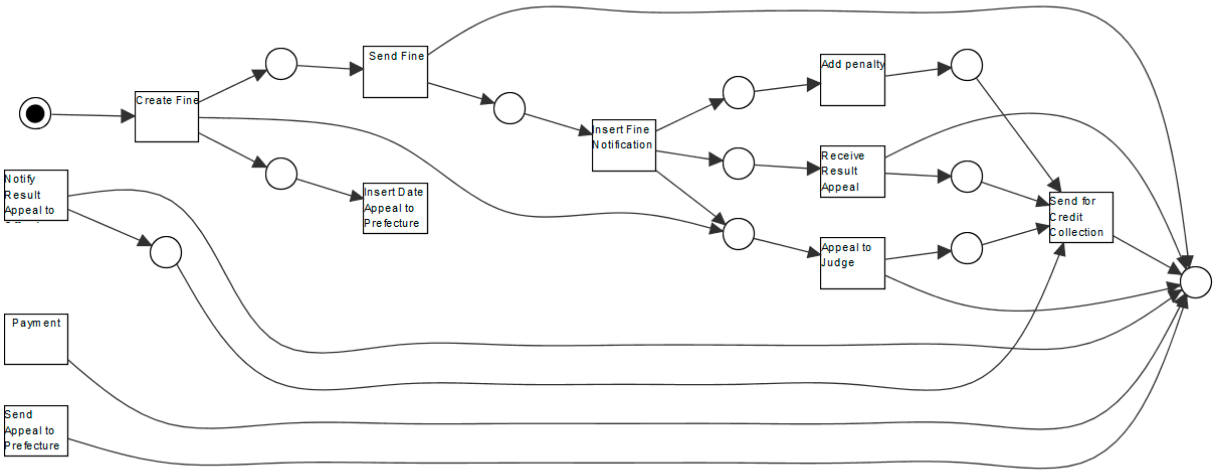

Figura 4.2: Red de Petri usando algoritmo α.

Como se puede observar, el modelo generado por el algoritmo α presenta varios inconvenientes. En primer lugar, existen transiciones sin conexión desde el inicio, como las actividades *Notify Result Appeal to Prefecture*, *Payment* y *Send Appeal to Prefecture*, que no están directamente vinculadas al flujo inicial del proceso y que se pueden ejecutar siempre. Esto implica que el modelo es incompleto y no refleja adecuadamente todas las posibles ejecuciones del sistema representado en el registro.

Además, se evidencia que no se han reflejado todas las trazas del log de eventos. Por ejemplo, la traza que hemos comentado anteriormente (crear multa, enviar multa, insertar notificación de multa, agregar penalización y pago) no está recogida en la red generada, porque, entre otras cosas, después de insertar notificación de multa, quedan habilitadas para dispararse, tanto agregar penalización como recibir resultado de apelación y apelación al juzgado. Este es un problema común del algoritmo α, el cual se basa únicamente en relaciones causales directas y no considera información más compleja, como bucles o elecciones condicionales.

Posteriormente, aplicando el algoritmo de minería inductiva al mismo registro de eventos, se genera el modelo de la figura 4.3.

En este caso, observamos que el modelo generado por el algoritmo de minería inductiva no tiene los problemas identificados en el modelo generado por el algoritmo α. En concreto, no hay transiciones desconectadas del lugar inicial y podemos observar que todas las trazas que aparecen en la figura 4.1 (que representan el 88,29 % de las trazas) son secuencias posibles de esta red de Petri.

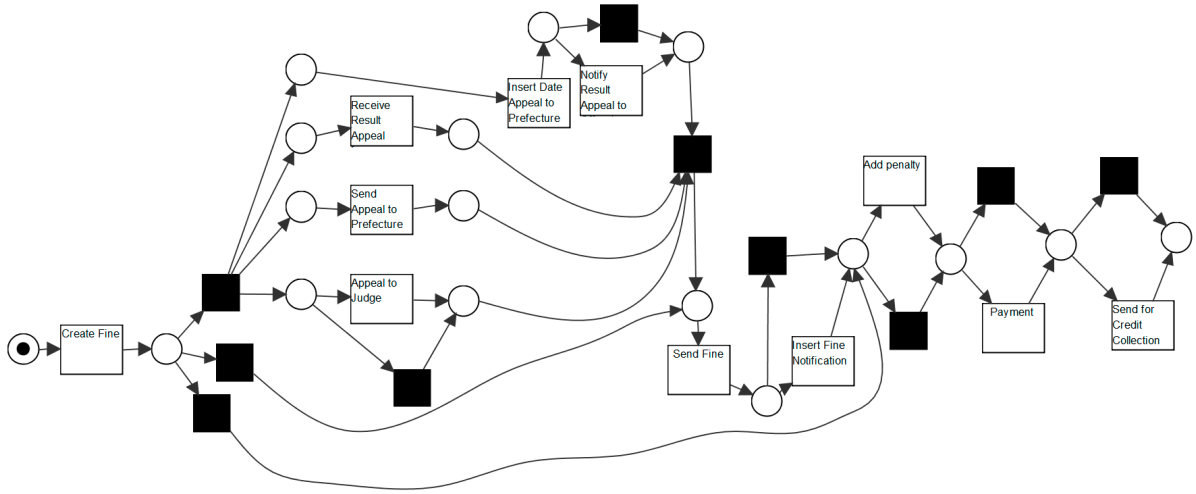

Figura 4.3: Red de Petri usando algoritmo minería inductiva.

Bibliografía

[1] VAN DER AALST (2016), *Process Mining: Data science in action.* 2.ª ed., Springer Berlin Heidelberg.

[2] VAN DER AALST, W. (2014) Process mining in the large: a tutorial. *Business Intelligence: Third European Summer School, eBISS 2013, Dagstuhl Castle, Germany, July 7-12, Tutorial Lectures 3, 2014,* pp. 33-76.

[3] VAN DER AALST, W. & WEIJTERS, AJMM (2005) Process mining. *Process-Aware Information Systems: Bridging People and Software through Process Technology,* John Wiley & Sons, Inc pp. 235-255.

[4] RIVAS, M. H., & BAYONA-ORÉ, S. (2019). Algoritmos de Minería de Proceso para el Descubrimiento Automático de Procesos. *Revista Ibérica de Sistemas e Tecnologias de Informação,* (31), pp. 33-49.

[5] VAN DER AALST, W. (2012). Process mining. Using real event data to X-ray business processes helps ensure conformance between design and reality. *Communications of the ACM,* 55(8) pp. 76-83.

[6] DE MEDEIROS, A. A., VAN DONGEN, B. F., VAN DER AALST, W. M., & WEIJTERS, A. J. M. (2004). Process mining: extending the alpha-algorithm to mine short loops. *BETA Working Paper Series*, vol. 113, pp. 145–180

[7] KÜSTERS, A., & VAN DER AALST, W. M. (2023, JUNE). Revisiting the Alpha Algorithm To Enable Real-Life Process Discovery Applications. *In ATAED/PN4TT@ Petri Nets.*

[8] DE MEDEIROS, A. K. A., VAN DER AALST, W. M., & WEIJTERS, A. J. M. M. (2003). Workflow mining: Current status and future directions. *In On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE: OTM, Catania, Sicily, Italy, November 3-7, 2003. Proceedings* (pp. 389-406). Springer Berlin Heidelberg.