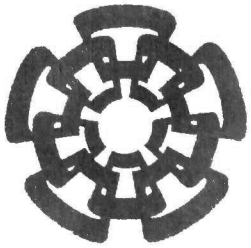
CT-788-SS1
Don.- 2014



Centro de Investigación y de Estudios Avanzados
del Instituto Politécnico Nacional
Unidad Guadalajara

Síntesis de Redes de Petri para Minería de Procesos Basada en la Inferencia de T-Invariantes

Tesis que presenta:
Tonatiuh Tapia Flores

para obtener el grado de:
Maestro en Ciencias

en la especialidad de:
Ingeniería Eléctrica

Director de Tesis
Dr Luis Ernesto López Mellado

CINVESTAV del IPN Unidad Guadalajara, Guadalajara, Jalisco, Agosto de 2013

Síntesis de Redes de Petri para Minería de Procesos Basada en la Inferencia de T-Invariantes

Tesis de Maestría en Ciencias
Ingeniería Eléctrica

Por:
Tonatiuh Tapia Flores
Ingeniero en Sistemas Computacionales
Instituto Tecnologíco de Ocotlán 2007-2011

Becario de Conacyt, expediente no. 263566

Director de Tesis
Dr. Luis Ernesto López Mellado

CINVESTAV del IPN Unidad Guadalajara, Agosto de 2013

Síntesis de Redes de Petri para Minería de Procesos Basada en la Inferencia de T-Invariantes

Resumen

La minería de procesos es el problema de descubrir un modelo adecuado a partir de una secuencia de tareas que representan la ejecución de algún proceso de negocio: por lo regular esta información es tomada de un sistema de gestión de flujo de trabajo. En esta tesis se presenta un método para obtener sistemáticamente una red de Petri (RP) a salvo (1-acotada) a partir de una secuencia de tareas S formada por una gran cantidad de trazas de ejecución de procesos. El método está basado en la determinación relaciones causales y de concurrencia entre las tareas. Se propone una técnica para inferir los t-invariantes a partir de S, éstos permiten por un lado, definir la estructura inicial de una RP y por otro lado, ajustar el modelo cuando los t-invariantes determinados no coinciden con los del modelo inicial; en particular se logra descubrir dependencias implícitas entre transiciones que no fueron observadas consecutivamente. Los algoritmos derivados del método han sido implementados y probados con numerosos ejemplos de complejidad diversa.

Petri Net Synthesis for Processes Mining Based on Inference of T-Invariants

Abstract

Process mining is the problem of discovering a suitable model from sampled tasks sequences representing the execution of some business processes; usually this data is provided by a workflow management system. In this thesis a method for obtaining systematically a safe (1-bounded) Petri net (PN) form a tasks sequence S composed by numerous processes execution traces. The method is based on determining causal and concurrency relations between tasks. A technique for inferring the t-invariants from S is proposed; the determined invariants allow, on the one hand, determine the initial structure of a PN, and on the other hand, adjust the model when the inferred t-invariants do not coincide with that of the initial model; in particular we are able to discover implicit dependencies between transitions that have not been observed consecutively. The algorithms derived from the methods have been implemented and tested on numerous examples of diverse complexity.

Agradecimientos

Mi mas sincero agradecimiento

A mis padres y hermano por su apoyo.

A Erika Alejandra por su consejos y su paciencia.

A mis compañeros por compartir tantos momentos durante esta maestría.

A Ana Paula Estrada por compartir su conocimiento y experiencia en el tema.

A CINVESTAV por la oportunidad de realizar mi maestría en ciencias.

A CONACYT por la beca recibida durante el programa de maestría.

Al LURPA de la ENS de Cachan por permitirme trabajar en sus instalaciones durante una estancia de investigación.

Al Prof. Jean-Jacques Lesage por su apoyo y asesoramiento durante mi estancia.

Al Dr. Luis Ernesto López Mellado por su valioso asesoramiento durante el proyecto de tesis y por todo el conocimiento adquirido.

Introducción

El minado de procesos de flujo de trabajo, trata el problema de identificar un modelo que represente una secuencia de eventos, tal y como están siendo ejecutados en algún proceso, este es un tema de actual interés, el cual a tomado fuerza en la ultima década debido al gran esfuerzo que se realiza por conocer a detalle, el flujo de las tareas que se realizan dentro de un proceso de negocio. El minado de procesos de flujo de trabajo aporta a organizaciones con procesos de negocio complicados, conceptos, herramientas y técnicas que ayudan a la administración de estos procesos dentro de la organización.

La primera vez que se aborda el problema de identificar determinadas características acerca de una secuencia de elementos pertenecientes a un alfabeto, fue hacia la inferencia gramatical como se puede observar en trabajos de [Gold, 1967] donde un autómata finito es construido a partir de muestras correctas de palabras aceptadas y el trabajo de [Angluin, 1987] por su parte muestra un método de aprendizaje que obtiene un autómata finito que reproduce las palabras aceptadas por cierto lenguaje .

Existen muchos métodos en la literatura, para la identificación de sistemas de eventos discretos (DES), uno de ellos y del cual se han retomado algunas definiciones para el desarrollo del siguiente trabajo, es el trabajo de [Estrada, 2013] en el cual se presenta un método para construir modelos de redes de Petri interpretadas (RPI) a partir de observaciones de entrada salida de un sistema de eventos discretos, este consta de dos principales pasos, uno que construye la parte observable del modelo y otro que construye un modelo no observable a partir de una secuencia de disparo de transiciones: generada por la secuencia de entrada salida, en este trabajo se trata de mejorar y extender la segunda parte del trabajo antes mencionado.

Hablando de los diferentes métodos enfocados hacia la identificación de procesos de flujo de trabajo, encontramos en primera instancia el algoritmo Alfa propuesto en [van der Aalst, 2003], el cual usa un workflow log de varios workflow traces compuestos por tareas ordenadas secuencialmente, después bajo el supuesto de que el workflow log es completo, se construye una red de Petri llamada workflow net, la cual representa dicha secuencia de tareas. Varios trabajos han extendido este método incorporando nuevas complejidades a los modelos que es capas de encontrar, tal es el caso de los siguientes trabajos [van der Aalst, 2009], [Wang, 2011].

Un enfoque probabilístico para la identificación de procesos de workflow es propuesto en [Cook, 2004] la entrada a este método es una secuencia de tareas que representan las actividades de cierto proceso de flujo de trabajo, después con ayuda de un proceso probabilístico se encuentra las relaciones causales y concurrentes entre las tareas, para arrojar como resultado patrones individuales de comportamiento. En [Agrawal, 1988] se propone una identificación basada en grafos de relaciones entre tareas llamado (conformal graph) que reproduce una secuencia de tareas conforme sus dependencias y las ejecuciones pasadas presentes en la secuencia de tareas.

Uno de los métodos mas recientes es el trabajo de [Wang, 2012] en el cual se tratan las relaciones implícitas entre tareas, solo que como entrada toma un conjunto llamado (event multiset) que contiene información de las post-actividades de cada una de las actividades presentes en el proceso y se asume que existe un manejador de workflow capaz de obtener esta información, el método es una modificación al algoritmo Alfa mencionado anteriormente.

En la técnica propuesta en este trabajo se usa como entrada una secuencia de transiciones, la cual es tomada de la primera parte del método mostrado en [Estrada, 2013], de cualquier forma la técnica igual puede trabajar con otra información, por ejemplo un workflow log. El algoritmo utiliza alguna de las definiciones vistas en el trabajo previo, para detectar las relaciones causales y concurrentes además de extender las definiciones incorporando una técnica para detectar los t-invariantes del modelo de red de Petri (RP) final en base a las dependencias repetitivas entre tareas, con ayuda del conocimiento acerca de los t-invariantes una nueva técnica para construcción del modelo RP es propuesta, por ultimo el método es capaz de detectar dependencias implícitas entre las tareas presentando para ello dos técnicas que atacan diferentes problemas, Una herramienta de software para la identificación de modelos a partir de una secuencia de transiciones fue desarrollada y probada con una maquina industrial experimental.

El resto de esta tesis esta organizado de la siguiente manera.

- Capítulo 1 donde se presentan los diferentes métodos de identificación que fueron estudiados.
  Capítulo 2 se presenta la definición del problema y se muestra en resumen el trabajo de [Estrada, 2013] para poder explicar en el capítulo 3 el trabajo principal de esta tesis.
  Capítulo 3 se presenta un algoritmo para la identificación de procesos de flujo de trabajo, incluyendo el análisis de las principales propiedades y características, así como también algunos ejemplos a manera de comprender mejor cada parte del algoritmo.
  Capítulo 4 se describe de maneta general la herramienta de software realizada para probar el algoritmo propuesto.
  Finalmente, algunas conclusiones y prospectivas de este trabajo son presentadas

Capítulo 1

Métodos de Minería de Procesos de Eventos Discretos

Resumen. En este capítulo se examinan las diferentes técnicas de minería procesos para sistemas de eventos discretos más importantes encontradas en la literatura y se hace un análisis de los enfoques recientes que abordan este problema. Finalmente una comparación entre los diferentes enfoques aquí analizados es presentada.

1.1 Métodos derivados de la teoría de lenguajes

Los primeros trabajos que se interesaron en resolver el problema de identificación, fueron enfocados hacia el área de teoría de lenguajes, donde el problema es generar una descripción del lenguaje a partir de un conjunto palabras del mismo, por medio de técnicas de aprendizaje o bien inferencias sobre el lenguaje.

El método de Gold [Gold, 1967] es una técnica de aprendizaje que procesa muestras las cuales contienen todas las posibles cadenas del lenguaje a identificar, el método hace una conjetura en base a la información recibida y compara con el lenguaje a identificar su ejecución termina cuando ambos lenguajes son los mismos.

Mas tarde, basado en los conocimientos de su trabajo anterior, Gold [Gold, 1972], desarrolla un método para la identificación de sistemas de tipo caja negra (black-box), que crea una representación del espacio de estados en base a información de experimentos que se realizan con las muestras del lenguaje. Por último el resultado del método puede ser presentado como una máquina de Mealy o como una maquina de Moore.

El problema de identificar un lenguaje regular con ayuda de muestras que pertenecen y no pertenecen al lenguaje es tratado por Angluin [Angluin, 1987], donde con la ayuda de un *"Minimally Adequate Teacher"* (oráculo) el cual puede responder a las preguntas de membresía y equivalencia; el método obtiene un autómata que representa el lenguaje.

1.2 Métodos para la identificación de Sistemas de Eventos Discretos

En la ultima década los investigadores han propuesto muchos métodos para la identificación de sistemas de eventos discretos, los cuales utilizan formalismos tales como las redes de Petri o autómatas para representar el modelo de un sistema y todos sus posibles comportamientos.

1.2.1 Enfoques de identificación

Los métodos de identificación de sistemas de eventos discretos (DES) permiten la construcción sistemática de un modelo matemático (rede de Petri, autómata) que describe el comportamiento de un sistema desconocido o poco conocido basándose en la observación de su comportamiento. Estas observaciones consisten de datos que contienen la actividad del sistema; secuencias de operaciones, eventos, mensajes, señales etc., y estos modelos permiten la reproducción del comportamiento observado.

Existen varios enfoques para la identificación de sistemas de manufactura de eventos discretos que han sido propuestos en la literatura. El enfoque de síntesis incremental presentado por Meda [Meda, 2001], [Meda, 2005], trata con DES no conocidos parcialmente medibles que presentan comportamiento cíclico. Muchos algoritmos para construir RPI han sido propuestos permitiendo la identificación en línea de DES concurrentes a partir de secuencias de salida, a pesar de que las técnicas son eficientes, los modelos obtenidos pueden representar mas comportamientos que los que fueron observados.

Otro método reciente [Klein, 2005], permite la construcción de un eficiente y no determinístico FA(NFA) a partir de un conjunto de secuencias entrada-salida, medidas del DES a ser identificado. El NFA obtenido genera exactamente las secuencias entrada-salida (I/O) de una longitud dada que las que fueron observadas. El método fue usado para la

detección de faltas en [Roth, 2009] y extendido para obtener una partición optima de subsistemas concurrentes para la detección de faltas distribuida en [Roth, 2012].

Un enfoque fuera de línea basado en programación lineal entera (ILP) produce modelos free-labels RP que representan exactamente la información presentada en la secuencia de eventos [Guia, 2005].

1.2.2 Identificación entrada-salida

En este enfoque se basa en un controlador lógico programable (PLC) basado en un sistema automático. El objetivo es descubrir, a partir de las observaciones del comportamiento del sistema expresado como una solo secuencia de señales de entrada-salida, como los componentes del sistema están interrelacionados y construir un modelo consistente el cual pueda mostrar explícitamente el comportamiento descubierto, en particular, concurrencia, sincronización, recursos compartidos, etc. La identificación de sistemas en operación envuelve dos aspectos importantes a considerar: el propio sistema en operación y las observaciones del proceso. Ambas cuestiones deben ser consideradas en un método para la identificación entrada-salida.

A continuación se describe de manera abstracta un método [Estrada, 2010] para analizar la secuencia de estrada-salida observada y establecer una relación clara entre las salidas y entradas del controlador. el método propuesto permite la construcción de un modelo reducido que representa la parte observable del modelo que se produce como consecuencia de la secuencia de entrada-salida, un modelo completo RPI. Cabe mencionar que ninguno de los métodos vistos en la literatura es capaz de reproducir un modelo bien estructurado.

1.2.3 Identificación caja-negra de DES Automatizados

El problema a resolver en este trabajo es construir el modelo para un DES a partir de una secuencia finita de vectores eventos de estrada-salida, recolectada durante la ejecución del sistema en un tiempo determinado [Estrada, 2010]. El tipo de sistemas considerados en este trabajo consisten en una planta y su controlador industrial operado en lazo cerrado.

Este método construye una red de Petri interpretada que representa las muestras entrada-salida obtenidas. Se procesa la secuencia de vectores de entrada salida para generar una secuencia de eventos y una secuencia de sub-cadenas, cada sub-cadena es representada por una transición en la red de Petri; esto describe la relación causal entre los eventos. La secuencia completa de eventos es representada por lugares no observables en la red de Petri. Por último el modelo es completado agregando lugares observables ligados a las transiciones de acuerdo a las cambios observados en los eventos de salida.

1.2.4 Método de identificación estadístico

Este método mejora notablemente el trabajo realizado por A. Estrada en 2010 ya que construye una red de Petri interpretada (RPI), la cual describe las relaciones causales y concurrentes observadas entre los eventos basándose en un enfoque estadístico el cual que permite el análisis de grandes secuencias de entrada-salida [Estrada, 2012]. El método consta de dos pasos generales un que construye la parte observable de la Red y otro que determina los lugares no observables y la relación entre los eventos.

Algoritmo

Antes de presentar el algoritmo es necesario explicar algunos conceptos tomados de [Estrada, 2012]. Primero se definen los tipos de eventos. La secuencia de vectores de eventos observados es derivada de la palabra de entada-salida w, $E(j) = w(j+1) - w(j)$; cada uno de estos ventores se divide en vectores de entrada y de salida.

$$E(j) = \begin{bmatrix} IE(j) \\ OE(j) \end{bmatrix}, \text{ donde } IE(j) = \begin{bmatrix} IE_0(j) \\ IE_1(j) \\ \vdots \\ IE_{m-1}(j) \end{bmatrix} \text{ y } OE(j) = \begin{bmatrix} OE_0(j) \\ OE_1(j) \\ \vdots \\ OE_{m-1}(j) \end{bmatrix}$$

De estos se determinan los eventos elementales los cuales se definen de la siguiente manera

$$IE(j) = IE_{j1} \quad IE_{j2} \ldots = \prod IE_{ji} \text{ t.q. } I_{ji}(j+1) - I_{ji}(j) \neq 0$$

$$OE(j) = OE_{j1} \bullet OE_{j2} \ldots = \prod OE_{ji} \text{ t.q. } O_{ji}(j+1) - O_{ji}(j) \neq 0$$

Si ningún evento curre en $E(j)$, se denota como

$$IE(j)=\varepsilon \ (OE(j)=\varepsilon).$$

De manera similar se puede representar cada vector de entrada

$$I(j) = I_{j1} \cdot I_{j2} \bullet \ldots = \prod I_{ji} \text{ t.q } I_{ji} = \begin{cases} I_i = 1 \, if \, I_i(j) = 1 \\ I_i = 0 \, if \, I_i(j) = 0 \end{cases}$$

1.3 Métodos de minería de procesos

Los modelos de procesos de negocio han ido ganando popularidad a medida que los sistemas de información cada vez generan más datos para su organización. Es importante estudiar algunos de los métodos para la minería de procesos ya que estos son muy parecidos a la identificación de sistemas de eventos discretos.

1.3.2 Minería de procesos por grafo dirigido

Los sistemas de flujos de trabajo son procesos los cuales pueden verse como la ejecución de acciones unitarias llamadas actividades, entre estas actividades existen diferentes dependencias. Un método presentado por Rakesh Agrawal [Agrawal, 1998], identifica las diferencias entre estas actividades y presenta un grafo que define el workflow del cual son tomadas muestras de palabras.

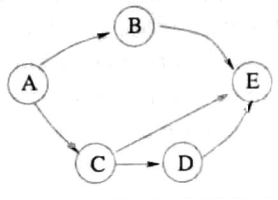


Figura 1.1: Ejemplo de flujo de trabajo

Algoritmo

Para explicar de manera clara el funcionamiento del algortimo es necesario tomar algunos conceptos definidos en [Agrawal, 1998], El registro de ejecución (log) de un proceso es una simple secuencia de eventos que inicia y termina con el mismo evento, estos eventos son llamados START y END respectivamente, *ABCE, ACDBE, ACDE* son ejemplos de registros de ejecución para el workflow de la figura 1.

Dado un log de la ejecución de un proceso se dice que la actividad B *sigue* a la actividad A, si la actividad B comienza después de que la actividad A termina. Si B *sigue* a la actividad A pero A no *sigue* ala actividad B se dice entonces que B *depende* de A. Entonces en el siguiente log de ejecuciones de un proceso determinado *{ABCE, ACDE, ADBE}* ; B sigue A y B depende de A. El *grafo de dependencia* de un conjunto de actividades V y un log de ejecuciones L, es aquel que tiene un camino de la actividad u a la actividad v si y solo si, v depende de u.

Algoritmo 1.1 Construcción del grafo de dependencias

Entrada: *L* (Log de ejecuciones).

Salida: *G* (Grafo obtenido).

1. Inicialmente el grafo $G = (V,E)$ con V como el conjunto de actividades y $E = 0$.
2. Para cada proceso d ejecución en L, y para cada par de actividades u, v tal que v depende de u agregar un arco (u, v) a E.
3. Remover de E aquellos arcos que aparezcan en ambas direcciones.
4. Por cada componente fuertemente conexa en G. Remover de E todos los arcos entre vértices en la misma componente fuertemente conexa.
5. Para cada proceso de ejecución en L (a) Encontrar el sub-grafo incluido en G. (b) Computar la reducción transitiva del sub-grafo (c) marcar aquellos arcos en E que están presentes en la reducción transitiva.
6. Remover los arcos que permanecieron desmarcados en E.
7. En el grafo obtenido unir los vértices que pertenecen a la misma instancia de una actividad.

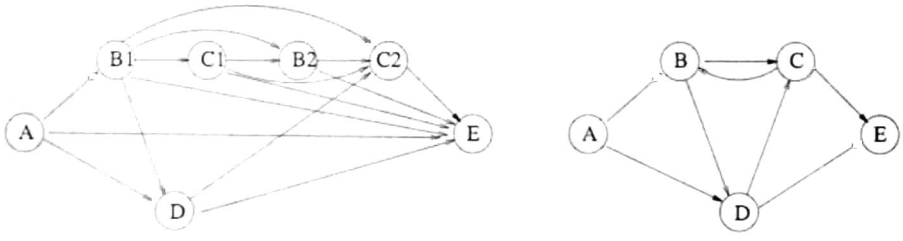


Figura 1.2 Ejemplo

Ejemplo. Considere el siguiente log L = *{ABDCE, ABDCBCE, ABCBDCE, ADE};* el grafo de la izquierda figura 1.2 es el grafo computado después del paso 4 del algoritmo. El Grafo resultante después de hacer el paso 7 es el grafo de la parte derecha de la figura 1.2.

1.3.2 Minería de procesos por dependencia/ frecuencia

La siguiente técnica crea un grafo de dependencia para determinar el flujo de trabajo de una secuencia de eventos generada por un proceso [van der Aalst, 2004]. En general el

método utiliza 3 pasos; 1) Construcción de una tabla de dependencia frecuencia denominada (D/F-table) 2) Inducción de un grafo dependencia/frecuencia (D/F-graph) a partir de la D/F-table y por ultimo 3) Reconstrucción del flujo de trabajo a partir de los datos obtenidos.

La D/F-table debe ser llenada con la siguiente información (i) la frecuencia total del evento (#A), (ii) la frecuencia de una tarea A cuando precede directamente de una tarea B (B<A), (iii) la frecuencia de una tarea A cuando se sigue de otra tarea B (A>B), (iv) la frecuencia de una tarea A directa o indirectamente dependiente por otra tarea B pero antes de la siguiente aparición de A (B<<<A), (v) la frecuencia de A cuando sigue directa o indirectamente de B pero antes de la siguiente aparición A (A>>>B) y por ultimo (vi) una métrica para evaluar la certeza de una relación causal entre A y otra tarea B (A→B). Ésta última depende de un valor $\delta = 0.8$ el cual fue determinado empíricamente. La causalidad (A→B) se incremente con un factor de $(\delta)^n$ donde n es el numero de eventos intermediarios entre la tarea A y la tarea B.

Para la *inducción del grafo de dependencia frecuencia* se usan reglas de heurística. La primera de ellas bajo esta observación.

$$R1: SI((A \rightarrow B \geq N)\ Y\ (A>B \geq \sigma)\ Y\ (B<A \leq \sigma))\ ENTONCES <A,B> \in T$$

Donde N es un factor de ruido al cual se le asigna un valor default de 0.05, y σ es calculado utilizando la siguiente ecuación $\sigma = 1 + Round\ (N*\#L/\#T)$, donde N es el factor de ruido #L es el número de ejecuciones del workflow y #T es el número de tareas diferentes. A manera de mejorar los resultados obtenidos con la regla *R1* es extendida con las siguientes dos reglas

$$R2: SI((A \rightarrow A \approx 0)\ Y\ (A<A + A>A\ > 0.5*\#A)\ Y\ (A<A - A>A \approx 0))\ ENTONCES <A,A> \in T$$

$$R3: SI((A \rightarrow B \approx 0)\ Y\ (A>B \geq \sigma)\ Y\ (B<A \approx A>B)\ Y\ (A>>>B \geq 0.4 * \#A)\ Y(B<<<A \approx A>>>B))\ ENTONCES <A,B> \in T$$

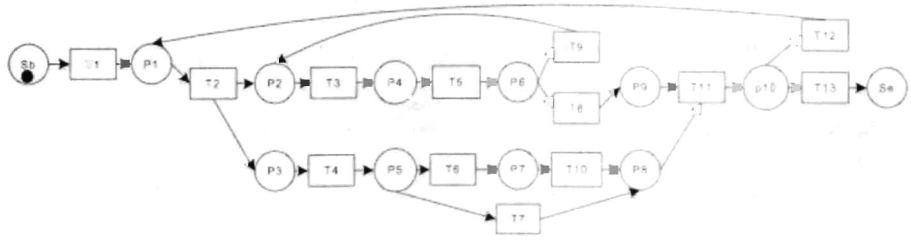


Figura 1.3 modelo de workflow representado como una red de Petri

Ejemplo. Dado el proceso de workflow de la figura 1.3, un registro de 1000 secuencias de eventos es tomado para generar los resultados que se muestran en la tabla 1.1. Con esta información se obtiene el grafo de dependencia/frecuencia de la figura 1.4. Por ultimo el proceso de generar la workflow net modelada como red de Petri dado el grafo de dependencia frecuencia es relativamente sencillo.

	#B	B<A	A>B	B<<<A	A>>>B	A→B
T10	1035	0	581	348	1035	0.803
T5	3949	80	168	677	897	0.267
T11	1994	0	0	528	1035	0.193
T13	1000	0	0	0	687	0.162
T9	1955	50	46	366	538	0.161
T8	1994	68	31	560	925	0.119
T3	3949	146	209	831	808	0.019
T6	1035	0	0	348	348	0.000
T7	959	0	0	264	241	-0.011
T12	994	0	0	528	505	-0.093
T1	1000	0	0	687	0	-0.246
T2	1994	0	0	1035	505	-0.487
T4	1994	691	0	1035	505	-0.825

Tabla 1.1 ejemplo de tabla-D/F para la tarea T6.

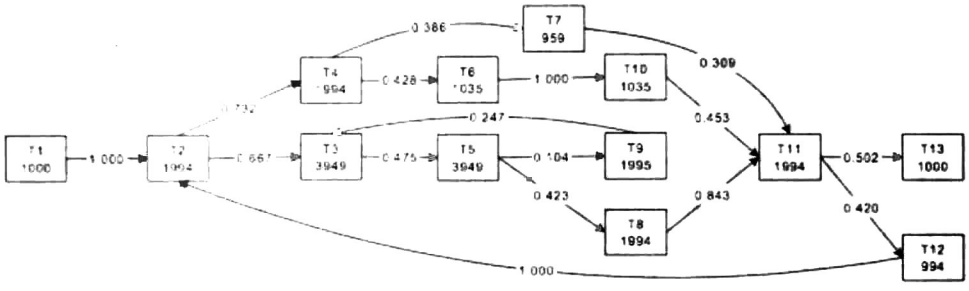


Figura 1.4 DF-graph del ejemplo

1.3.3 Algoritmo Alfa

En [van der Aalst, 2004] se ataca el problema de identificar una red workflow basándose en la información de varios registros de ejecución de la red, los cuales tienen varias tareas que se registran de manera secuencial. Estos registros de ejecución son los datos de entrada para el algoritmo.

Antes de presentar el algoritmo se incluye la notación usada por van der Aalst para definir las dependencias entre las distintas tareas del workflow.

Sea T un conjunto de actividades. $\sigma \in T^*$ es una secuencia workflow y $W \in \mathcal{P}(T^*)$ es un registro de workflow.

Sea W un registro sobre T ,si $a, b \in T$:

- $a >_W b$ ssi existe $\sigma = t_1 t_2 t_3 \ldots t_{n-1}$ y $i \in \{1,\ldots,n\text{-}2\}$ t.q. $\sigma \in W$ and $t_i = a$ y $t_{i+1} = b$,
- $a \rightarrow_W b$ ssi $a >_W b$ y $b \not>_W a$,
- $a \#_W b$ ssi $a \not>_W b$ y $b \not>_W a$,
- $a \|_W b$ ssi $a >_W b$ y $b >_W a$.

Sea A un conjunto, $a \in A$, y $\sigma = a_1 a_2 \ldots a_n \in A^*$ una secuencia sobre A de longitud n:

$a \in \sigma$ ssi $a \in \{a_1, a_2, \ldots, a_n\}$,
$first(\sigma) = a_1$, si $n \geq 1$,
$last(\sigma) = a_n$, si $n \geq 1$.

Algoritmo 1.2 Algoritmo Alfa

Entrada: σ (Event traces), T (tareas)

Salida: $\alpha(W) = (P_W, T_W, F_W)$.

1. $T_W = \{t \in T \mid \exists \sigma \in W, t \in \sigma\}$
2. $T_I = \{t \in T \mid \exists \sigma \in W, t = first(\sigma)\}$
3. $T_O = \{t \in T \mid \exists \sigma \in W, t = last(\sigma)\}$
4. $X_W = \{(A,B) \mid A \subseteq T_W \wedge B \subseteq T_W$
   $\wedge \forall a \in A \forall b \in B, a \rightarrow_W b$
   $\wedge \forall a_1, a_2 \in A, a_1 \#_W a_2$
   $\wedge \forall b_1, b_2 \in B, b_1 \#_W b_2\}$
5. $Y_W = \{(A,B) \in X_W \mid \forall (A',B') \in X_W, A \subseteq A'$
   $\wedge B \subseteq B' \Rightarrow (A,B) = (A',B')\}$
6. $P_W = \{p(A,B) \mid (A,B) \in Y_W\} \cup \{i_W, o_W\}$
7. $F_W = \{(a, p_{(A,B)}) \mid (A,B) \in Y_W \wedge a \in A\}$
   $\cup (p_{(A,B)}, b) \mid (A,B) \in Y_W \wedge b \in B\}$
   $\cup (i_W, t) \mid t \in T_I\} \cup (t, o_W) \mid t \in T_O\}$

Ejemplo. Dado el registro mostrado en la tabla 1.2

Case identifier	*Task identifier*
Case 1	*Task A*
Case 2	*Task A*
Case 1	*Task B*
Case 1	*Task C*
Case 2	*Task C*
Case 3	*Task A*
Case 2	*Task B*
Case 2	*Task D*
Case 4	*Task A*
Case 3	*Task C*
Case 1	*Task D*
Case 3	*Task B*
Case 4	*Task E*
Case 4	*Task D*
Case 3	*Task D*

Tabla 1.2 Registro de workflow.

Del cual puede deducirse la siguiente traza de eventos $W = \{ABCD, ACBD, AED\}$, el resultado de computar el algoritmo α es el siguiente.

1. $T_W = \{A, B, C, D, E\}$
2. $T_I = \{A\}$
3. $T_O = \{D\}$
4. $$X_W = \{(\{A\},\{B\}),(\{A\},\{C\}),(\{A\},\{E\}),$$
$$(\{B\},\{D\}),(\{C\},\{D\}),(\{E\},\{D\}),$$
$$(\{A\},\{B,E\}),(\{A\},\{C,E\}),(\{B,E\},\{D\}),$$
$$(\{C,E\},\{D\})\}$$
5. $$Y_W = \{(\{A\},\{B,E\}),(\{A\},\{C,E\}),$$
$$(\{B,E\},\{D\}),(\{C,E\},\{D\})\}$$
6. $$P_W = \{i_W, o_W, p_{(\{A\},\{B,E\})}, p_{(\{A\},\{C,E\})}$$
$$p_{(\{B,E\},\{D\})}, p_{(\{C,E\},\{D\})}\}$$
7. $$F_W = \{(i_W, A),(A, p_{(\{A\},\{B,E\})})$$
$$(p_{(\{A\},\{B,E\})}), B..., (D, o_W)\}$$
8. $\alpha(W) = (P_W, T_W, F_W)$.

El resultado puede observarse de manera más clara en la workflow net de la figura 1.5

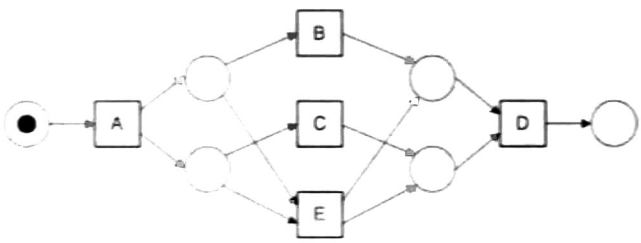


Figura 1.5 Modelo resultante para el registro del ejemplo 1

1.3.4 Algoritmo Lambda.

Este trabajo es una mejora al anteriormente presentado y gran parte de la notación expuesta en el trabajo de [van der Aalst, 2004]. Su principal ventaja es que evita la obtención de los registros de tareas de la traza de tareas. En [Wang, 2012] se propone el algoritmo λ el cual en general, recupera un multiconjunto de eventos *MS* que contiene la frecuencia de cada evento ignorando sus ordenes de aparición, esto con ayuda de información de las post-actividades; después con esta información obtiene las relaciones entre las actividades. El problema con este planteamiento es que se tiene la hipótesis de que existe un manejador de workflow que es capaz de obtener las post-actividades de cada actividad.

A manera de explicar el método tomaremos de [Wang, 2012] la definición de un multiconjunto de eventos *MS*. Supongamos que la información que nos muestra el manejador de workflow es la que se muestra en la siguiente tabla, entonces el multiconjunto de eventos para este registro es el siguiente $MS = \{A[CD], B^2[CE], C[D], D[], C^2[E], E^2[]\}$.

Nombre de la tarea	*Post-tareas*	*Número*
A	*C,D*	*1*
B	*C,E*	*2*
C	*D*	*1*
D		*1*
C	*E*	*2*
E		*2*

Tabla 1.3 Registro de eventos con post actividades

La siguiente notación es usada en el algoritmo, *m(e)* obtiene el número de eventos en *MS, task(e)* obtiene el nombre de la tarea del evento *e* y *post(e)* obtiene la post-actividad del evento *e*. La función *CalEventNum* se usa en el algoritmo para contar el número de ocurrencias de un evento que su tarea está en el conjunto de tareas y su post actividad contiene post actividades. Si tomamos el registro de la figura 1.4 entonces CalEventNum({A}, C) + CalEventNum({B}, C) = CalEventNum({A, B},C) = CalEvent Num({C}, θ) = 3, y CalEventNum({B}, E) = CalEventNum({C}, E) = CalEventNum({E}, θ) = 2.

Algoritmo 1.3 Algoritmo λ

Entrada: σ (Event traces), *T* (tareas)

Salida: $\alpha(W) = (P_W, T_W, F_W)$.

1. $T_W = \{t \in T \mid \exists e \in MS, t = task(e)\}$,
2. $T_I = \{t \in T \mid t \in first(MS)\}$
3. $T_O = \{t \in T \mid t \in last(MS)\}$
4. $$X_W = \{(PS,SS) \mid PS \subseteq T_W \wedge SS \subseteq T_W \wedge \forall a \in PS \forall b \in SS, a \rightarrow_W b \wedge \forall a_1, a_2 \in PS, a_1 \#_W a_2 \wedge \forall b_1, b_2 \in SS, b_1 \#_W b_2 \wedge \forall b \in SS, \quad CalEventNum(MS,PS,b) = CalEventNum(MS,PS,\phi)\},$$
5. $$Y_W = \{< PS,SS > \in X_W \mid \forall < PS',SS' > \in X_W, PS \subseteq PS' \wedge SS \subseteq SS' \Rightarrow < PS,SS > = < PS',SS' >\},$$
6. $P_W = \{p < PS,SS > \mid < PS,SS > \in Y_W\} \cup \{i_W, o_W\}$
7.

$$F_W = \{(a, p < PS, SS >) \mid (PS, SS) \in Y_W \wedge a \in PS\}$$
$$\cup (p < PS, SS >, b) \mid (PS, SS) \in Y_W \wedge b \in SS\}$$
$$\cup (i_w, t) \mid t \in T_I\} \cup (t, o_W) \mid t \in T_O\}, \quad y$$

Ejemplo. Considerando que el multiconjunto de eventos arrojado generado a partir de un sistema es el siguiente

$$MS = \{t_1^8[t_2], t_2^8[t_4 t_9], t_4^4[t_6 t_8], t_6^{16}[t_8], t_8^8[t_9], t_9^8[t_{11}], t_6^8[t_7], t_7^8[t_6], t_4^{12}[t_5, t_6]$$
$$, t_5^{12}[t_8], t_5^3[t_5], t_1^8[t_3], t_3^8[t_4, t_{10}], t_8^8[t_{10}], t_{10}^8[t_{11}], t_{11}^{16}[]\}$$

El resultado de aplicar el algoritmo λ es el siguiente.

1. $T_W = \{t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8, t_9, t_{10}, t_{11}\}$,
2. $T_I = \{t_1\}$
3. $T_O = \{t_{11}\}$
4. $X_W = \{(\{t_1\}, \{t_2\}), (\{t_1\}, \{t_3\}), (\{t_1\}, \{t_2, t_3\}), (\{t_2, t_3\}, \{t_4\}), (\{t_2\}, \{t_9\}),$
   $(\{t_3\}, \{t_{10}\}), (\{t_4, t_5\}, \{t_8\}), (\{t_4, t_5\}, \{t_5, t_8\}), (\{t_6\}, \{t_7\}), (\{t_6\}, \{t_8\}),$
   $(\{t_6\}, \{t_7, t_8\}), (\{t_4, t_7\}, \{t_6\}), (\{t_8\}, \{t_9\}), (\{t_8\}, \{t_{10}\}), (\{t_8\}, \{t_9, t_{10}\}),$
   $(\{t_9, t_{10}\}, \{t_{11}\})\}$,
5. $Y_W = \{(\{t_1\}, \{t_2, t_3\}), (\{t_2\}, \{t_9\}), (\{t_3\}, \{t_{10}\}), (\{t_2, t_3\}, \{t_4\}), (\{t_4, t_5\}, \{t_5, t_8\}),$
   $, (\{t_4, t_7\}, \{t_6\}), (\{t_6\}, \{t_7, t_8\}), (\{t_8\}, \{t_9, t_{10}\}), (\{t_9, t_{10}\}, \{t_{11}\})\}$,
6. $P_W = \{i_w, o_w, p(\{t_1\}, \{t_2, t_3\}), p(\{t_2\}, \{t_9\}), p(\{t_3\}, \{t_{10}\}), p(\{t_2, t_3\}, \{t_4\}), p(\{t_4, t_5\}, \{t_5, t_8\}),$
   $p(\{t, p(\{t_4, t_7\}, \{t_6\}), p(\{t_6\}, \{t_7, t_8\}), p(\{t_8\}, \{t_9, t_{10}\}), p(\{t_9, t_{10}\}, \{t_{11}\})\}$,
7. $F_W = \{(i_w, t_1), (t_1, p(\{t_1\}, \{t_2, t_3\})), (p(\{t_1\}, \{t_2, t_3\}), t_2), ..., (t_{11}, o_w)\}$,
8. $\lambda(W) = (P_W, T_W, F_W)$ *Como se muestra en figura* 1.6

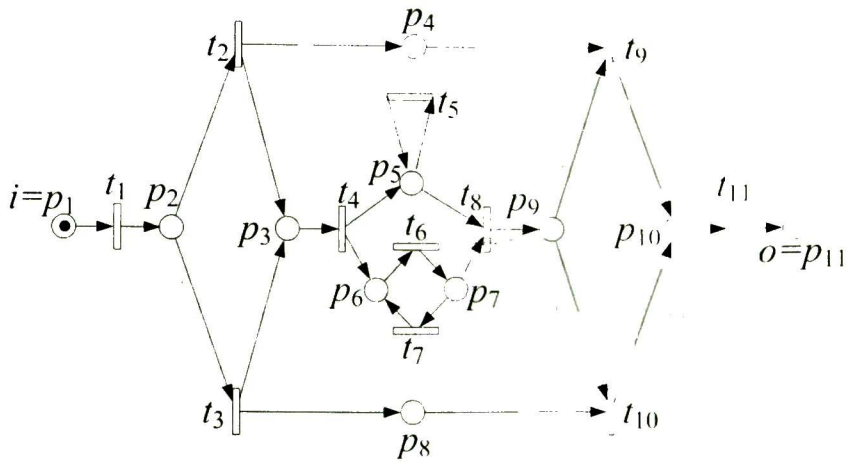


Figura 1.6 Ejemplo de modelo del proceso en red de Ptri

1.3.5 Minería de procesos con presencia de dependencia implícita

Al igual que el trabajo anterior en el presentado por [Lijie Wen, 2006], se usan los mismos razonamientos expuestos en [van der Aalst, 2004]. En este trabajo se presenta una técnica que puede tratar las dependencias implícitas que pueden observarse en un registro de eventos, para eso se exponen algunas reglas nuevas, tres teoremas y experimentos que prueban sus resultados.

El orden de las relaciones que existen entre las tareas es definido de la siguiente manera, sea W un registro de eventos sobre T . para $a, b \in T$

$a\ \Delta_W\ b$ ssi existe un camino $\sigma = t_1t_2t_3\ldots t_{n-1}$ y $i \in \{1,\ldots,n\text{-}2\}$ t.q. $\sigma \in W$ y $t_i = t_{i+2} = a$ and $t_{i+1} = b$,

$a >_W b$ ssi existe un camino $\sigma = t_1t_2t_3\ldots t_{n-1}$ y $i \in \{1,\ldots,n\text{-}2\}$ t.q. $\sigma \in W$ y $t_i = a$ and $t_{i+1} = b$,

$a \rightarrow_W b$ ssi $a >_W b$ y $b \not>_W a$ ó $a\ \Delta_W\ b$ ó $b\ \Delta_W\ a$

$a\ \#_W\ b$ ssi $a \not>_W b$ and $b \not>_W a$,

$a\ \|_W\ b$ ssi $a >_W b$ and $b >_W a$ and $\neg$ (a Δ_W b ó b Δ_W a),

$a \lhd_W b$ ssi $a\ \#_W\ b$ y existe una tarea c t.q. c $\in T$ y $c \rightarrow_W a$ y $c \rightarrow_W b$,

$a \rhd_W b$ ssi $a\ \#_W\ b$ y existe una tarea c t.q. c $\in T$ y $a \rightarrow_W c$ y $b \rightarrow_W c$,

$a >>_W b$ ssi $a \not>_W b$ y existe un camino $\sigma = t_1t_2t_3\ldots t_n$ y $i, j \in \{1,\ldots,n\}$ t.q. $\sigma \in W$ y $i < j$ y $t_i = a$ y $t_j = b$ y para cualquier $k \in \{i+1,\ldots,j\text{-}1\}$ se satisface que $t_k \neq a$, $t_k \neq b$ y $\neg(t_k \lhd_W a$ ó $t_k \rhd_W a)$,

$a\ a \succ_W b$ ssi $a \rightarrow_W b$ ó $a >>_W b$, y

$a \mapsto_W b$ ssi b tiene una dependencia implícita con a, o existe una dependencia implícita de b hacia a.

Además de las relaciones entre las tareas se definen los siguientes teoremas para tratar con las dependencias implícitas.

Teorema 1. Sea W un registro de tareas libre de auto lazos sobre T. $T_W = \{t \in T | \exists \sigma \in_W t \in \sigma\}$. Para cualquier tarea t t.q. $t \in T_W$. Si existen dos tareas $t_1\ t_2$ t.q. $t_1, t_2 \in T_W$ y $t_1 \rightarrow_W t$, $t_2 \rightarrow_W t$ y $t_1\ \#_W\ t_2$ y $\{t_1\}^* \neq \{t_2\}^*$, , calcular X_w, Y_w de la siguiente manera:

$X_W = \{(A,B) | A \subseteq T_w \; y \; B \subseteq T_w \; y \forall_a \in_A \forall_b \in_B a \rightarrow_W b \; y \; \forall_{a1,a2} \in_A a_1 \#_W a_2$
$y \; \forall_{b1,b2} \in_B b_1 \#_W b_2\}, \; y$
$Y_W = \{(A, B) \in X_W \mid \forall_{(A',B')} \in_{Xw} A \subseteq A', B \subseteq B' \rightarrow (A,B) = (A',B')\}.$

Teorema 2. Sea W un registro de tareas libre de auto lazos sobre *T*. $T_W = \{t \in T \mid \exists \sigma \in_W t \in \sigma\}$.

(i)Para cualquier tarea *t* t.q. $t \in T_W$. Si existen dos tareas $t_1 t_2$ t.q. $t_1, t_2 \in T_W$ y $t \rightarrow_W t_1$, $t \rightarrow_W t_2$ y $t_1 \|_W t_2$, , calcular X_w, Y_w de la siguiente manera:

$X_W = \{X \subseteq T_w \mid \forall_x \in_X t \rightarrow_W x \; y \; \forall_{x1,x2} \in_X x_1 \#_W x_2 \}$ y
$Y_W = \{X \in X_W \mid \forall_{X'} \in_{Xw} X \subseteq X' \rightarrow X = X'\}.$

(ii)Para cualquier tarea *t* t.q. $t \in T_W$. Si existen dos tareas $t_1 t_2$ t.q. $t_1, t_2 \in T_W$ y $t_1 \rightarrow_W t$, $t_2 \rightarrow_W t$ y $t_1 \|_W t_2$, , calcular X_w, Y_w de la siguiente manera:

$X_W = \{X \subseteq T_w \mid \forall_x \in_X x \rightarrow_W t \; y \; \forall_{x1,x2} \in_X x_1 \#_W x_2 \}$ y
$Y_W = \{X \in X_W \mid \forall_{X'} \in_{Xw} X \subseteq X' \rightarrow X = X'\}.$

Teorema 3. Sea W un registro de tareas libre de auto lazos sobre *T*. $T_W = \{t \in T \mid \exists \sigma \in_W t \in \sigma\}$. Para cualquiera tareas *a y b* t.q. $a, b \in T_W$, y $a \rhd_W b$, calcular X_w, Y_w de la siguiente manera:

$X_W = \{(A,B) | A \subseteq T_w \; y \; B \subseteq T_w \; y (\forall_{ai} \in_A a >>_W ai$ y $b \neg \gg_w$ ai) y $(\forall_{bj} \in_B a \neg \gg_W bj \; y \; b \gg$
bj) $y \; \forall_{ai} \in_A \exists_{bj} \in_B b_j \lhd_W a_i \; y \; \forall_{bj} \in_B \exists_{ai} \in_A a_i \lhd_W b_j \; y \; \forall_{ai,aj} \in_A a_i \|_W a_j \; y \; \forall_{bi,bj} \in_B b_i \|_W b_j\}, \; y$
$Y_W = \{(A, B) \in X_W \mid \forall_{(A',B')} \in_{Xw} A \subseteq A', B \subseteq B' \rightarrow (A,B) = (A',B')\}.$

La figura 1.7(a) muestra el resultado de aplicar el algoritmo α a un registro de tareas la red resultante es muy parecida a la expuesta en (b) excepto por los dos arcos punteados. Los cuales resultan de aplicar el Teorema 1, de donde se puede concluir $A \mapsto_W C$. Por último se pueden unir los lugares p1 y p2 para presentar la red de forma más clara.

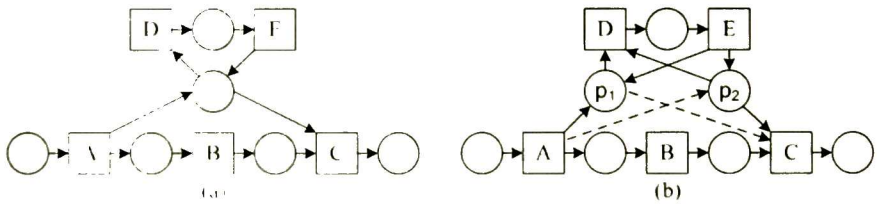


Figura 1.7 Dependencias implícitas encontradas aplicando Teorema 1.

1.3.6 Minería de procesos enfoque probabilístico

En el trabajo propuesto en [Cook, 2004] se plantea el problema de identificar patrones de comportamiento entre los eventos, los cuales pueden ser útiles para comprender el comportamiento completo del sistema. Se hace uso de un análisis probabilístico y estadístico para determinar tales patrones de comportamiento, esta técnica toma como entrada una secuencia de eventos y su objetivo principal es encontrar comportamientos individuales sobre la red.

Algoritmo

Antes de presentar el algoritmo es necesario tomar algunas definiciones que se usan en [Cook, 2004] aplicadas sobre la secuencia de eventos.

Prefix(S) es la secuencia *S* con su ultimo elemento removido en caso de que la secuencia tenga longitud 1 entonces *Prefix(S)* = null.
S:e es la secuencia S concatenada con el evento *e*.

- *Occur(S)* es el numero de ocurrencias de *S*, por exemplo *Occur(AC)* = 2 en CABCBACABCABCBAC. *Occur(null)* es el numero total de eventos en la traza de eventos.
- La probabilidad condicional de ocurrencia de un secuencia es:

$$\text{CondProb}(S) = P(S) = \frac{Occur(S)}{Occur(\text{Prefix}(S))}$$

- *Entropy,* da una medida de aleatoriedad entre las tareas y esta dada por la siguiente formula:

$$Entropy(S) = -\sum_{e \in E} P(S:e) x \log_{|E|}(P(S:e))$$

La cerradura de la entropia es un valor cercano a $log_n(T)$. El valor más cercano a este indica un bifurcacion entre last areas, puede usarse para detectar una possible unión si se analiza la secuencia en sentido contrario.
Event type counts: medida para determinar si el comportamiento de las actividades sucesoras a alguna actividad es concurrente o secuencial.
Si existe selección

- $Occur(S) = \sum_{e \in E, Occur(S:e)>0} Occur(e)$

Si existe concurrencia

$$\forall (e \in E, Occur(S:e) > 0), Occur(e) = Occur(S)\ T \times Occur(S) = \sum_{e \in E, Occur(S\ e)>0} Occur(e)$$

Causalidad: una vez que no se detecto que dos eventos como bifurcaciones o puntos de sincronizacion la siguiente regla ayuda a decidir si se trata de dos eventos causalmente dependientes o no.

- If P(AB) + P(BA) ≥ 1.5, son eventos causalmente dependientes.
- If P(AB) + P(BA) < 1.5, son eventos independientes.

Periodicidad: es una medida para detectar puntos de sincronizacion en el porceso position(S,i) es la posicion del ultimo evento de la ith aparicion de S, de aqui el periodo promedio de una secuencia S es.

$$PeriodMean(S) = \frac{\sum_{i=2}^{Occur(S)} Position(S,i) - Position(S,i-1)}{Occur(S) - 1}$$

Los eventos con una desviación estándar pequeña deberán ser marcados como puntos de sincronizacion en el proceso.

$$\text{PeriodStdDev}(S) = \sqrt{\frac{\sum_{i=2}^{\text{Occur}(S)} [\text{Dsq}(S,i) - \text{PeriodMean}(S)^2]}{\text{Occur}(S) - 2}}$$

Basándose en estas métricas se encuentra una medida de ranking *Rank(S)* que indica la calidad de la informacion recabada hasta el momento. Ejemplo Considere la red de Petri de la figura 1.8.

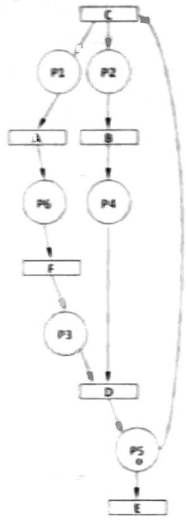


Figura 1.8 red de Petri

A partir de una secuencia de 1666 eventos producidos de una simulación estadística de la red la table 1.3 es deducida.

	A	B	C	D	E	F
A	0.00	0.25	0.00	0.00	0.00	0.75
B	0.44	0.00	0.00	0.31	0.00	0.25
C	0.56	0.44	0.00	0.00	0.00	0.00
D	0.00	0.00	0.99	0.00	0.01	0.00
E	0.00	0.00	0.00	0.00	0.00	0.00
F	0.00	0.31	0.00	0.69	0.00	0.00

Tabla 1.4 Probabilidades condicionales para cadenas de longitud 2.

Después de realizar el análisis del rango para casa evento, el método arroja como resultado el modelo de la figura 1.9.

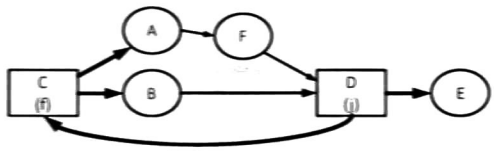


Figura 1.9 Modelo resultante

Este trabajo presenta una técnica probabilística para inferir modelos de sistemas con comportamientos concurrentes, dicha técnica presenta la necesidad de cadenas completas ya que secuencias pequeñas podrían nunca presentar eventos secuenciales, que realmente

tienen una relación en el sistema, ya que este es esencialmente un algoritmo voraz que nunca se retracta sobre las decisiones que toma sobre dependencias entre las tareas.

1.3 Caracterización de los métodos analizados

A manera de comparar nuestros resultados con los trabajos expuestos en este capítulo, se listan algunos de los problemas mas comunes en el área de minería de procesos, algunos de estos expuestos por van der Aalst en [van der Aalst 2004a].

Ruido: la información recolectada puede estar incorrecta o incompleta, generando problemas cuando al momento de hacer la minería.

Contriciones de tipo (Non-free choice): Decisiones de control que dependen de otras decisiones tomadas durante el proceso.

Minería de ciclos: El proceso puede tener varios ciclos de ejecución los cuales pueden tener una o mas tareas.

Diferentes Perspectivas: El método puede aplicarse a otras áreas además de resolver el problema de minería de procesos.

Visualizar resultados: El resultado de la minería de procesos puede ser representado de manera grafica mediante un modelo formal.

- *Procesos Concurrentes:* puede tratar con tareas que se ejecutan al mismo tiempo.
- *Dependencia implícita:* El método de minería de procesos es capaz de encontrar dependencias implícitas entre las tareas.

Las características principales de los métodos antes mencionados son resumidas en la tabla 1.5.

Criterio de comparación / Técnica de Process Mining	Ruido	Minería de ciclos	Diferentes perspectivas	Visualizar resultados	Procesos concurrentes	Dependencia implícita
[Agrawal, 1998]		X		X	X	
[Cook, 2004]				X	X	
[van der Aalst, 2004] algoritmo alfa				X		
[van der Aalst, 2005]	X	X		X	X	
[van der Aalst, 2004] dependencia/ frecuencia				X		
[Wang, 2012]		X		X	X	X
[Wang, 2006]		X		X	X	X
[APL, 2013]						

Tabla 1.5. Comparativa entre técnicas de minado de procesos.

Capítulo 2

Minería de Procesos de Flujo de Trabajo

Resumen. En este capítulo se define el problema de identificación de procesos en flujos de trabajo. En particular se describe un método de base anteriormente propuesto [Estrada, 2013] orientado a la identificación de sistemas de eventos discretos; de este trabajo se analizan sus características y limitantes. Finalmente se describe en forma general la propuesta de esta tesis.

2.1 Planteamiento del Problema

Por lo regular los manejadores de flujo de trabajo que actualmente existen en el mercado, necesitan de un modelo de proceso específico para poder operar. El objetivo entonces de la minería de procesos de flujo de trabajo es obtener, de un conjunto de tareas realizadas en el proceso, un modelo que las represente tal cual están siendo ejecutadas por dicho proceso.

Para poder realizar el modelo de flujo de trabajo, es necesario saber la secuencia de tareas que fueron realizadas en el proceso: esta información está contenida en el "*workflow log*". La mayoría de las técnicas analizadas en el capítulo 1 presentan la información del workflow log por medio de secuencias σ_i denominadas "*event trace*", en las que existe uno o más eventos que comienzan y terminan la ejecución. La tabla 2.1 muestra un workflow log dividido en casos.

Identificador	Secuencias de tareas
σ_1	A B C D
σ_2	A C B D
σ_3	A B C D
σ_4	A C B D
σ_5	A E D

Tabla 2.1. Workflow log dividido en casos.

El objetivo principal de la minería de flujo de trabajo es poder representar en un modelo formal la información representada en un Workflow log. Por ejemplo para el caso del workflow log de la tabla 2.1, la minería debería ser capaz de arrojar como resultado un modelo como el mostrado en la figura 2.1.

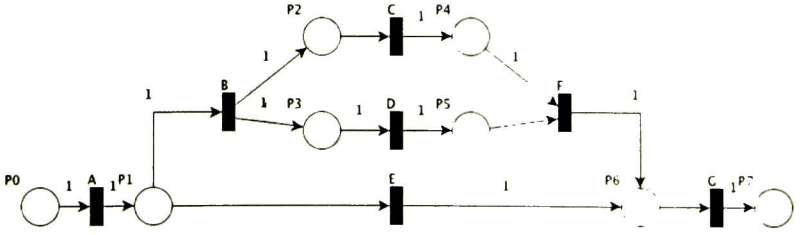


Figura 2.1. Modelo deducido de la tabla 2.1.

2.2 Trabajo previo.

En el trabajo de A. Estrada [Estrada, 2013] el problema abordado es la identificación de sistemas de eventos discretos, el cual es un problema similar al de minería de procesos. Los procesos a descubrir son sistemas compuestos por un controlador (PLC) y una planta los cuales trabajan bajo lazo cerrado son considerados, la figura 2.2 muestra a grandes rasgos la identificación del sistema desde el punto de vista del PLC.

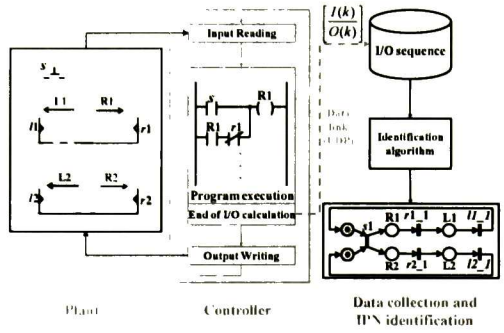


Figura 2.2 PLC + Planta y proceso de identificación[APE, 2012]

Como puede observarse en la imagen, el sistema tiene señales de entrada que son el resultado de las salidas de la planta y señales de salida que pueden verse como la siguiente entrada para la planta. A pesar de que en un sistema existen muchas señales de entrada y salida, la información que se usa para el proceso de identificación es una secuencia de vectores I/O w (2.1), donde $I(j)$ y $O(j)$ son respectivamente los valores de r entradas y q salidas en el ciclo del PLC. Para analizar la evolución de las señales es necesario trabajar con vectores de eventos los cuales son el resultado de la diferencia entre dos vectores I/O consecutivos $E(k) = w(k+1) - w(k) \neq 0$ (2.2); éstos a su vez pueden ser descompuestos como vectores de eventos entrada salida.

$$w = \left[\frac{I(1)}{O(1)}\right], \left[\frac{I(2)}{O(2)}\right], \left[\frac{I(3)}{O(3)}\right], \ldots \quad (2.1) \qquad E(k) = \left[\frac{IE(k)}{OE(k)}\right] \quad (2.2)$$

Se describen tres tipos de situaciones para describir el cambio observado en los vectores de eventos:

- $IE(j) \neq 0$ y $OE(j) \neq 0$: *Un cambio en la entrada provoca directamente un cambio a la salida;*
- $IE(j) = 0$ and $OE(j) \neq 0$: *El controlador llega en el evento j-1 a un estado en el cual, dados los valores de entrada, una evolución en la salida es observada en el evento j.*
- $IE(j) \neq 0$ and $OE(j) = 0$: *El controlador no es sensible a un cambio en la entrada. O provoca un cambio de estado que no es observado en un cambio a la salida.*

Dada la información recabada hasta el momento el proceso de identificación se lleva a cabo en dos pasos.

Paso 1: Descubrir el comportamiento reactivo de entradas y salidas, donde la parte observable de la red es construida como pequeños fragmentos de red, los cuales tienen como etiqueta de sus transiciones, expresiones algebraicas que indican las variables de entrada del sistema.

Paso 2: Inferir la parte no observable de la red, para ello la secuencia w es transformada en una secuencia S de disparos de las transiciones observables.

Esta segunda parte del método de identificación es muy importante para nuestro trabajo, ya que a partir de este momento el problema puede verse como un problema de minería de flujo de trabajo, donde el objetivo es encontrar el comportamiento no observable del sistema a partir de un registro de eventos S. Abundaremos más sobre este segundo paso

después de un ejemplo

Antes de continuar con la explicación a detalle de la segunda parte del método es necesario introducir la definición formal de red de Petri la cual se presenta en la siguiente definición.

Definición 2.1. Una estructura de red de Petri ordinaria denominada G es un grafo bipartita dirigido representado por la cuádrupla $G = (P, T, I, O)$ donde: $P = \{p_1, p_2, ..., p_{|P|}\}$ y $T = \{t_1, t_2, ..., t_{|T|}\}$ conjuntos finitos de vértices llamados lugares y transiciones respectivamente; $I(O) : P \times T \rightarrow \{0,1\}$ es una función que representa los arcos que van de una transición a un lugar o bien de un lugar a una transición.

La matriz de incidencia de G denotada por $C = C^+ - C^-$, donde $C^- = [c_{ij}^-]$; $c_{ij}^- = I(p_i, t_j)$; y $C^+ = [c_{ij}^+]$; $c_{ij}^+ = O(p_i, t_j)$ son el pre y el post de la matriz de incidencia respectivamente.

La función de marcado $M : P \rightarrow \mathbb{Z}^+$ representa el numero de marcas que se encuentran en cada lugar; el cual es usualmente representado con el vector $|P|$-entrada. $\mathbb{Z}^+$ es el conjunto de enteros no negativos.

Definición 2.2: Un sistema de red de Petri o simplemente res de Petri (RP) se define como $N = (G, M_0)$, donde G es una estructura RP y M_0 es un marcado inicial dado.

En un sistema de red de Petri, una transición t_j es habilitada para el marcado M_k si $\forall p_i \in P$, $M_k(p_i) \geq I(p_i, t_j)$; una transición habilitada t_j puede ser disparada generando un nuevo marcado M_{k+1}. Este comportamiento es representado por la función $M_k \xrightarrow{t_j} M_{k+1}$ y puede computarse como $M_{k+1} = M_k + Cv_k$, donde $v_k(i) = 0$, $i \neq j$, $v_k(j) = 1$; Esta ecuación es llamada la ecuación de estado de un sistema de red de petri. El conjunto de alcanzabilidad de una RP es el conjunto de marcados alcanzables a partir del marcado inicial M_0 con el disparo solo transiciones habilitadas; este conjunto es representado por $R(G, M_0)$.

Ejemplo 2.1 Considere el sistema de manufactura mostrado en la figura 2.3 [Estrada, 2012], cuyo objetivo es ordenar paquetes de acuerdo a su tamaño, este tiene 9 entradas; señales generadas por sensores para detectar la posición de los cilindros (a_0, a_1, a_2, b_0, b_1, c_0, c_1) y la presencia de los paquetes (k_1 ,k_2) y cuatro señales de salida que controlan los actuadores de la planta (A+, A-, B, C), la forma en que se presentan los vectores I/O es la siguiente $[A+\ A-\ B\ C\ k_1\ k_2\ a_0\ a_1\ a_2\ b_0\ b_1\ c_0\ c_1]^T$

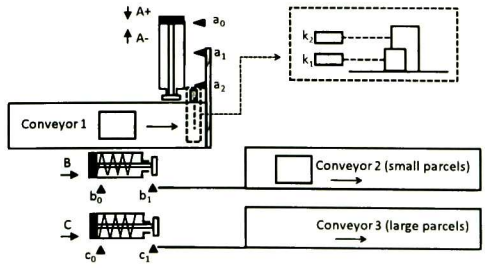


Figura 2.3 Sistema de manufactura Ejemplo 2.1

Una secuencia de 216 vectores I/O fue procesada de los cuales solo los primeros son presentados en la figura 2.4, aquí mismo también se muestra los fragmentos de RPI (para este ejemplo solo fue encontrado uno), las etiquetas para cada transición y la secuencia S obtenida de w.

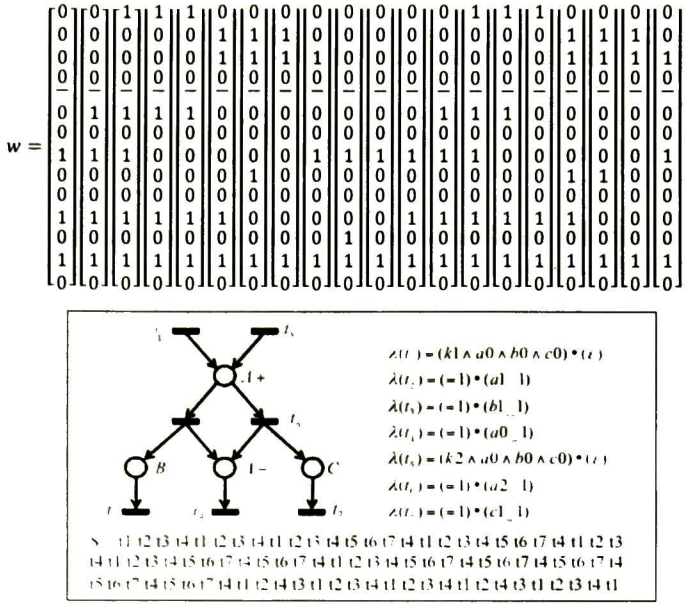


Figura 2.4 Resultados Ejemplo 2.1

Centrándonos en la segunda parte del método la cual determina la parte no observable del modelo, puede definirse el modelo como sigue, dada una RPI observable (P^{obs}, T, Pre^{obs} $Post^{obs}$) y una secuencia de transiciones $S = t_1\ t_2\ ...\ t_j\ ... \in T^*$ reproducida por la secuencia I/O w, una estructura de RP (P^{nobs}, T, Pre^{nobs} $Post^{nobs}$) que reproduce S y un marcado inicial M_0 que habilite S debe ser encontrado. La nueva estructura se define como N=(P, T, I, O) con P= $P^{obs} \cup P^{nobs}$, Pre= $Pre^{obs} \cup Pre^{nobs}$, $Post$= $Post^{obs} \cup Post^{nobs}$. El resultado deberá ser una red a salvo.

Esta técnica obtiene de S la relación de precedencia y concurrencia entre las transiciones. Para poder encontrar estas relaciones es necesario definir algunas propiedades para poder definir las relaciones causales y después algunas reglas para la construcción de la red.

Definición 2.3. Las relaciones entre transiciones en S que son observadas consecutivamente es expresada en la relación $Seq \subseteq T \times T$ que es definida como $Seq = \{(t_j, t_{j+1}) | \ 1 \leq j < |S| \ \}$. Si $(t_a, t_b) \in Seq$, se expresa como $t_a < t_b$.

Una vez que se encontraron las relaciones en *Seq* estas deben clasificarse por su tipo de comportamiento el cual puede caber en alguna de las clasificaciones de la siguiente definición.

Definición 2.4. Cada par de transiciones que fueron observadas consecutivas t_a, t_b en *Seq* puede ser clasificada en una de las siguientes situaciones

Relación causal. Si la ocurrencia de t_a habilita t_b. En una estructura RP, esto implica que existe por lo menos un lugar entre t_a y t_b.

Relación de concurrencia. Si ambas t_a y t_b son simultáneamente habilitadas, pero t_a ocurre primero y no deshabilita a t_b, en una estructura de RP, esto implica que es imposible

la existencia de un lugar de t_a a t_b. En este caso se dice que, t_a y t_b, son concurrentes y se denota como $t_a||t_b$.

La siguiente definición se refiere a la dependencia repetitiva que tiene un evento respecto a otro evento, es decir la ocurrencia de las transiciones necesarias para volver a observar cierta transición.

Definición 2.5. Una transición t_j depende repetitivamente de t_k, y se denota como $t_k \prec t_j$, si solo si t_k se observa en todos los casos entre dos apariciones de t_j en S. Por convención, diremos que $t_j \prec t_j$ siempre y cuando t_j se observe por lo menos dos veces en S. Por lo tanto el *conjunto de dependencias repetitivas* de una transición t_j estará dado por la función Rd: $T \rightarrow 2^T$, que indica cuales transiciones deben ser ejecutadas para rehabilitar el disparo de t_j, p.e. $Rd(t_j)=\{t_k \mid t_k \prec t_j\}$. Si t_j fue observada solo una vez en S, entonces $Rd(t_j) = \varnothing$.

Definicion 2.6. Dos transiciones t_a ,t_b están en un ciclo de longitud dos (2-c), si S contiene la subcadena $t_a t_b t_a$ o la subcadena $t_b t_a t_b$. el conjunto de transiciones en un 2-c es denotado por $Tc=\{(t_a,t_b) \mid t_a, t_b$ *están en un 2-c*$\}$.

A manera de entender de forma clara las definiciones hasta el momento consideremos el siguiente ejemplo.

Ejemplo 2.2. A partir de la secuencia $S = t_1\ t_2\ t_3\ t_4\ t_1\ t_2\ t_4\ t_3\ t_5\ t_6\ t_7\ t_4\ t_1\ t_2\ t_3\ t_4\ t_5\ t_6\ t_7\ t_4\ t_1\ t_2\ t_3\ t_4\ t_1\ t_2\ t_3\ t_4\ t_5\ t_6\ t_7\ t_4\ t_5\ t_6\ t_7\ t_4\ t_1\ t_2\ t_3\ t_4\ t_5\ t_6\ t_7\ t_4\ t_5\ t_6\ t_7\ t_4\ t_5\ t_6\ t_7\ t_4\ t_5\ t_6\ t_7\ t_4\ t_5\ t_6\ t_7\ t_4\ t_1\ t_2\ t_4\ t_3\ t_1\ t_2\ t_3\ t_4\ t_1\ t_2\ t_3\ t_4\ t_1\ t_2\ t_4\ t_3\ t_1\ t_2\ t_3\ t_4\ t_1$, se puede deducir que $t_2 \prec t_1$, $t_3 \prec t_1$, $t_4 \prec t_1$, por lo tanto. $Rd(t_1)=\{t_1, t_2, t_3, t_4\}$, y el resto de los conjuntos de dependencias repetitivas es el siguiente, $Rd(t_2)=\{t_1, t_2, t_3, t_4\}$, $Rd(t_3) = \{t_1, t_2, t_3\}$, $Rd(t_4) = \{t_4\}$, $Rd(t_5) = \{t_4, t_5, t_6, t_7\}$, $Rd(t_6) = \{t_4, t_5, t_6, t_7\}$, $Rd(t_7) = \{t_4, t_5, t_6, t_7\}$. Por último el conjunto de transiciones observadas consecutivamente es $Seq=\{(t_1,t_2), (t_2,t_3), (t_3,t_4), (t_4,t_1), (t_4,t_5), (t_5,t_6), (t_6,t_7), (t_7,t_4), (t_2,t_4), (t_4,t_3), (t_3,t_1)\}$. Para este ejemplo ningún par de transiciones presentó comportamiento de 2-c por lo tanto $Tc = \varnothing$.

Definición 2.7. Un circuito en una RP es un camino que comienza y termina en el mismo nodo. Un circuito se dice que es *simple*, si este no usa la misma transición más de una vez, y *elemental* si no usa el mismo lugar más de una vez.

Después de haber definido estas propiedades podemos ahora clasificar las relaciones que encontramos en *Seq* según sus características en relaciones causales o relaciones concurrentes.

2.2.1 Relaciones causales.

Proposición 2.1. Si existe la relación $t_a \prec t_b$ entonces, debe existir en N un circuito simple y elemental (circuito SE) al cual ambas transiciones t_a, t_b pertenecen.

Proposición 2.2. Si existe la relación $t_a \prec t_b$ y $t_a \prec t_b$ entonces, debe existir en N un lugar de t_a a t_b.

Proposición 2.3. Si $(t_a,t_b) \in Tc$, entonces debe de existir en N un lugar de t_a a t_b y un lugar de t_b a t_a.

Se puede notar que si dos transiciones fueron vistas en S consecutivamente y además una es dependiente repetitiva de la otra entonces existe una relación causal entre estas

transiciones al igual que para las transiciones que están en un 2-c. Para clasificar estas relaciones se define a continuación el conjunto de relaciones causales *CausalR*.

Definición 2.8. El conjunto de relaciones causales *CausalR* tiene registro de todos los pares de transiciones que fueron definidas como causales en S.

$$CausalR = \{(t_a,t_b) \mid (t_a<t_b) \text{ and } (t_a \prec t_b \text{ or } t_b \prec t_a \text{ or } (t_a,t_b) \in Tc)\}.$$

Ejemplo 2.3. Para la información obtenida anteriormente del ejemplo 2.2 podemos obtener el conjunto de relaciones causales $CausalR = \{(t_1,t_2), (t_2,t_3), (t_4,t_1), (t_4,t_5), (t_5,t_6), (t_6,t_7), (t_7,t_4), (t_2,t_4), (t_3,t_1)\}$.

Como se dijo antes, para dos transiciones que se encuentren en *CausalR*, debe existir entre ellas un lugar que las conecte, esto para preservar el orden de aparición de las transiciones en S. A continuación se explicarán los casos en los que la aparición de dichos lugares puede omitirse.

2.2.2 Relaciones concurrentes

En el caso en que dos transiciones t_a , t_b se encuentren en una relación concurrente, entonces no debe existir la presencia de un lugar entre éstas, sea de t_a a t_b o bien de t_b a t_a ya que si esto llegara a pasar la ejecución de una deshabilitaría el disparo de la otra.

Definición 2.9. Dos transiciones t_a , t_b las cuales fueron observadas consecutivamente en ambas direcciones es decir $(t_a, t_b) \in Seq$, $(t_b, t_a) \in Seq$ y no están en *Tc* en una secuencia *completa* de eventos *S* , pueden definirse como transiciones concurrentes denotadas como $(t_a \| t_b)$. El conjunto de transiciones que fueron observadas concurrentes es llamado, *ConcR*, éste se define: $ConcR = \{(t_a, t_b) \mid t_a < t_b \wedge t_b < t_a \wedge \nexists (t_a, t_b) \in Tc\}$.

Si la secuencia de eventos *w* es completa, es decir que muestra todas los posibles comportamientos del sistema observado, entonces podremos descubrir todas a aquellas relaciones de concurrencia entre las tareas.

Proposición 2.4. Sean t_a t_b dos transiciones las cuales han sido observadas consecutivas en una secuencia completa *S* en ambos sentidos p. ej. $(t_a , t_b) \in Seq$, $(t_b , t_a) \in Seq$. Entonces $(t_a , t_b) \notin CausalR$ y $(t_b , t_a) \notin CausalR$ ssi $t_a \| t_b$.

2.2.3 Construir la parte no observable de la red

Las relaciones causales y concurrentes encontradas a partir de la secuencia *S* pueden usarse para inferir la evolución interna del sistema, calculando los lugares no observables de la red.

Definición 2.10. El conjunto $Seq' = ((Seq \setminus CausalR) \setminus ConcR) \setminus ConcR^{-1}$ contiene todos los pares de transiciones que fueron observados consecutivamente, pero no pudieron definirse como causales o concurrentes; donde $ConcR^{-1} = \{(t_b, t_a) \mid (t_a, t_b) \in ConcR\}$.

En caso de que $Seq' \neq \emptyset$, existen dos posibilidades para estas dos transiciones.

De ahora en adelante un lugar que será añadido para indicar la relación causal entre t_a y t_b será denotado por $[t_a , t_b]$. Cuando un lugar tiene más de una transición de entrada o de salida puede denotarse como $[t_{a1}\ t_{a2} \ldots t_{al} \cdot t_{b1}\ t_{b2} \ldots t_{bh}]$, donde $l = |\cdot p|$ y $h = |p \cdot|$. Si la transición t_k es observada seguida de dos transiciones $t_{ai}\ t_{aj}$ en *S*, existen dos casos para representar este comportamiento; el primero de ellos cuando t_a t_b nunca fueron vistas concurrentes, entonces estas relaciones pueden ser representadas por el mismo lugar: $[t_k, t_{ai}$

t_{aj}], para el caso en el que existe concurrencia entonces cada relación tiene su propio lugar: [t_k,t_{ai}] [t_k, t_{aj}].

En la figura. 2.5 los lugares observables [t_2, t_3] y [t_6, t_7] existen en la red. Agregando los lugares no observables al modelo [t_2, t_3] y [t_6, t_7], podemos obtener la RPI que se muestra en la Figura 2.5.b, la cual reproduce la secuencia *w*.

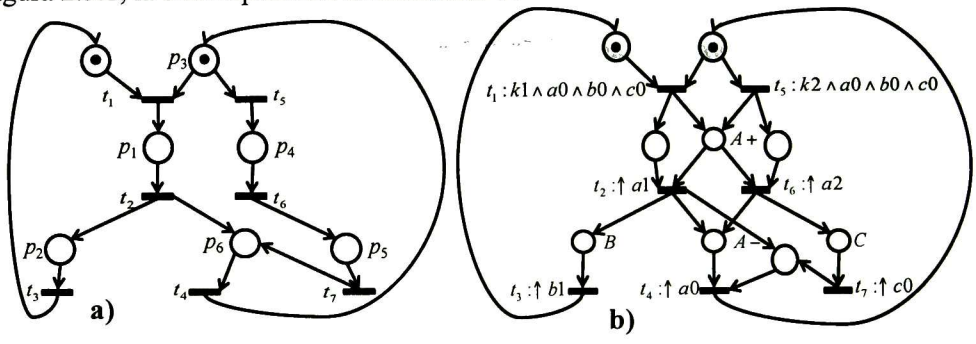


Figura 2.5 a) Modelo no observable; b) Modelo final identificado

2.2.4 Verificación de flujo de marcas

La secuencia *w* podría no mostrar la suficiente combinación entre transiciones para detectar todas las concurrencias. A manera de aproximar lo más posible el lenguaje de la RP a *S*, se considera que si dos transiciones no han sido declaradas como concurrentes, serán declaradas como relaciones secuenciales. Sin embargo, si las transiciones son concurrentes, la secuencialidad podría llevarnos a colocar arcos y lugares que interfieran con la ejecución de *S*. Para verificar que los lugares agregados no interfieran con la correcta ejecución de *S*, se presenta una regla de corrección para los lugares agregados.

Proposición 2.11. En el modelo RPI construido hasta el momento, cada lugar no observable p en N debe cumplir con la siguiente ecuación de entrada salida.

$$\sum_{t_i \in \cdot p} Occ(t_i) = \sum_{t_i \in p^\cdot} Occ(t_i) \pm 1$$

Donde Occ(t_i) es el número de ocurrencias de t_i en *S*.

Proposición 2.10 Si existe un lugar *p* t.q. $|{}^\cdot p|=1$, entonces $\forall t_j \in {}^\cdot p$, $t_k \in Rd(t_j)$, $Rd(t_j)\neq\varnothing$ donde t_k es la transición de entrada de *p*. También, si existe un lugar *p* t.q. $|p^\cdot|=1$, entonces $\forall t_j \in p^\cdot$, $t_k \in Rd(t_j)$, $Rd(t_j)\neq\varnothing$ donde t_k es la transición de salida de *p*.

Regla de corrección. Si la ecuación de entrada salida o las condiciones en la proposición 2.10 no se satisfacen para algún lugar, los arcos relacionados a éste que no estén en el conjunto *CausalR* son removidos; en el caso en que no existan relaciones causales para este lugar el lugar es removido.

Ejemplo 2.5 Para la secuencia *S* = t3 t5 t4 t8 t1 t2 t7 t3 t2 t1 t4 t8 t7 t6 t5 t8 t5 t3 t4 t8 t5 t3 t8 t2 t3 t2 t1 t5 t8 t5 t6 t4 t1 t5 t6 t5 t6 t7 t4 t8 t7 t8 t1 t5 t6 t7 t4 t8 t3 t7 t4 t3 t8 t4 t5 t8 t5 t1 t8 t7 t2 t6 t1 t4 t3 t5 t8 t5 t8 t2 t7 t1 t6 t2 t1 t7 t8 t4 t7 t1 t2 t3 t6 t2 t1 t2 t5 t3 t8 t7 t4 t6 t3 t7 t8 t5 t2 t1 t2 t6 t1 t7 t6 t7 t8 t5 t4 t6 t7 t3 t2 t6 t3 t7 t4 t8 t5 t3 t4 t8 t5 t1 t6 t5 t6 t7 t6 t5 t8 t5 t6 t5 t8 t5 t8 t2 t3 t2 t1 t7 t4 t8 t1 t4 t5 t3 t6 t5 t2 t6 t1 t4 t1 t4 t1 t7 t2 t6 t5 t3 t6 t7 t2 t1 t6 t2 t5 t8 t7 t6 t5 t6 t1 t5 t2 t3 t2 t3 t8 t2 t1 t4 t3 t5 t8

t7 t4 t1 t8 t7 t2 t1 t2 t3 t6 t4 t7 t8 t7 t6, el modelo de red de Petri obtenido se muestra en la figura 2.6, la cual muestra dos redes de Petri independientes.

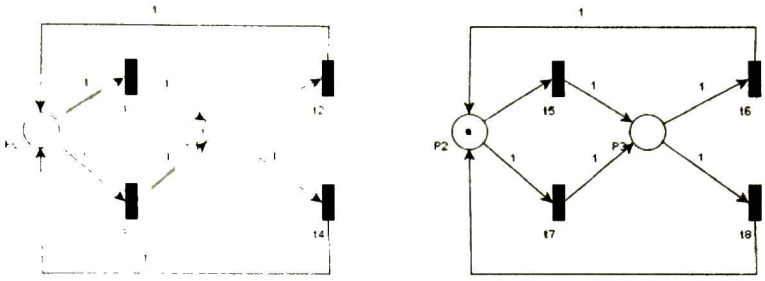


Figura 2.6 redes de Petri resultantes del ejemplo 2.5

2.3 Limitaciones del trabajo previo

A pesar de que la segunda parte del método visto en [Etrada, 2013] es muy completa existen algunos detalles que se extenderán y mejorarán en el método propuesto a lo largo del capítulo 3. A continuación se explican algunos de las situaciones en las que el algoritmo antes presentado no es capaz de obtener un modelo que reproduzca exactamente la secuencia de disparos de transiciones.

2.3.1 Encontrar las dependencias repetitivas reales

Como se explicó en la definición 2.3, la *Dependencia Repetitiva* de una transición t_i contienen el conjunto de transiciones que ocurren siempre repetitivamente para poder observar a t_i nuevamente; dicho de otra forma este conjunto de transiciones es parte de un invariante del sistema. Desgraciadamente la definición 2.3 no es suficiente para caracterizar todos los invariantes del sistema.

Ejemplo 2.6 Se tomara una simulación en PIPE de una secuencia de disparo para red de Petri de la figura 2.7 solo para ejemplificar algunas propiedades.

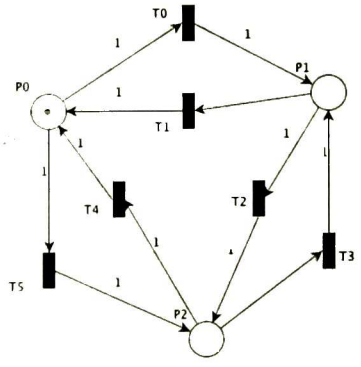


Figura 2.7 red de Petri para el ejemplo 2.6

La secuencia S = T5 T4 T5 T4 T5 T3 T1 T5 T3 T2 T4 T0 T2 T3 T1 T0 T2 T3 T2 T3 T1 T5 T4 T5 T4 T5 T3 T1 T5 T3 T2 T4 T0 T1 T0 T1 T0 T2 T3 T2 T4 T5 T3 T1 T0 T1 T5 T3 T1 T5 T4 T0 T1 T0 T1 T0 T2 T4 T5 T4 T0 T1 T5 T3 T2 T3 T2 T3 T1 T5 T4 T5 T3 T1 T0 T2 T4 T0 T2 T3 T2 T4 T0 T2 T3 T2 T3 T2 T4 T5 T4 T0 T1 T0 T2 T3 T1 T0 T2 T3 T2 T3

T1 T0 T2 T3 T2 T3 T1 T5 T4 T0 T2 T3 T1 T0 T2 T3 T2 T3 T2 T3 T2 T3 T2 T4 T5 T3 T2 T4 T0 T1 T5 T3 T1 T0 T1 T0 T1 T5 T4 T5 T3 T2 T4 T5 T4 T0 T2 T3 T2 T4 T0 T1 T0 T2 T3 T2 T3 T2 T4 T5 T3 T1..., fue tomada después de la simulación de donde se pueden calcular los siguientes conjuntos de dependencias repetitivas: *Rd(T0)={T0}*, *Rd(T1) = {T1}*, *Rd(T2) = {T2}*, *Rd(T3) = {T3}*, *Rd(T4) = {T4}* y *Rd(T5) = {T5}*, puesto que todos ellos contienen solo a ellos mismos hace falta más información para poder detectar los verdaderas componentes repetitivas del sistema, ya que podemos ver a simple vista que {T0, T1} forman una componente repetitiva; sin embargo ningún conjunto de *Rd* contiene estas transiciones.

En el siguiente capítulo se presentan algunas definiciones que extiendes la noción de dependencia repetitiva, permitiendo así detectar los invariantes del sistema lo que ayuda considerablemente al momento de construir el modelo final.

2.3.2 Relaciones implícitas en la secuencia S.

Algunos sistemas pueden presentar dependencias implícitas entre sus tareas, es decir tareas que dependen una de la aparición de la otra, pero nunca son observadas consecutivas en la secuencia *S* y por lo tanto no son tomadas en cuenta para la construcción del modelo final. En la mayoría de los casos se obtiene un modelo que se excede en su comportamiento, en comparación con el comportamiento presentado por la secuencia *S*. Veamos el siguiente ejemplo a manera de comprender mejor esta situación.

Ejemplo 2.7 La secuencia *S* = T2 T4 T5 T1 T2 T4 T5 T1 T2 T4 T5 T1 T3 T4 T6 T1 T3 T4 T6 T1 T3 T4 T6 T1 T3 T4 T6 T1 T3 T4 T6 T1 T3 T4 T6 T1 T2 T4 T5 T1 T2 T4 T5 T1 T2 T4 T5 T1 T3 T4 T6 T1 T2 T4 T5 T1 T3 T4 T6 T1 T3 T4 T6 T1 T2 T4 T5 T1 T2 T4 T5 T1 T2 T4 T5 T1 T3 T4 T6 T1 T2 T4 T5 T1 T2 T4 T5 T1 T2 T4 T5 T1 T3 T4 T6 T1 T2 T4 T5 T1..., fue tomada de la simulación en PIPE de una red de Petri, el resultado de aplicar la segunda parte del método visto en [Estrada, 2013] es mostrado en la figura 2.8 a) la parte b) muestra la red original de donde se hizo la simulación.

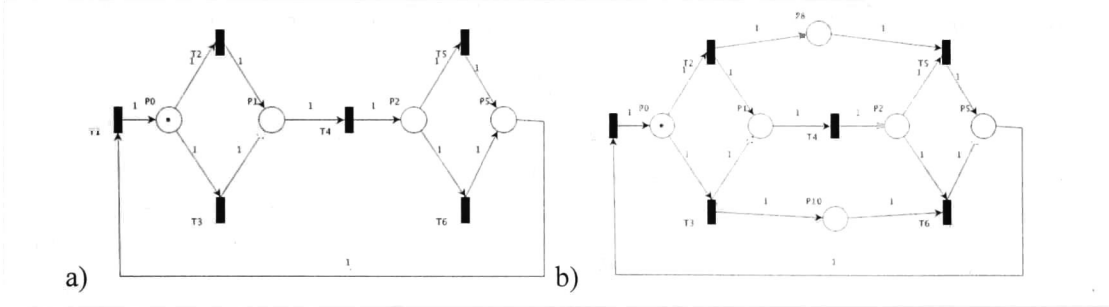


Figura 2.8. Petri net resultado del ejemplo 2.7

Como se observa en la figura 2.8.a, a diferencia de lo que muestra 2.8.b, hace falta la presencia de los dos lugares que unen T2 con T5 ([T2, T5]) y T3 con T6 ([T3, T6]), los que hace que sea un resultado erróneo ya que la red de la figura 2.8.a puede reproducir secuencias de tareas que no son permitidas en 2.8.b, como por ejemplo la secuencia T1 T2 T4 T6 o bien T1 T3 T4 T5. Con una simple inspección a la secuencia *S* nos percatamos de que las transiciones T2, T5 y T3 T6 nunca aparecen consecutivas, razón por la cual es imposible para el método detectar alguna dependencia entre ellas.

2.4 Minería de procesos vía Componentes Repetitivas visión general

El conocer información acerca del modelo final antes de tenerlo, es una de las principales ventajas que ofrece la minería de procesos vía componentes repetitivas; esto facilita la construcción del modelo y nos ofrece una alternativa para verificar la precisión de nuestro resultado. Antes de pasar al siguiente capítulo expondremos un resumen completo del método propuesto en este trabajo.

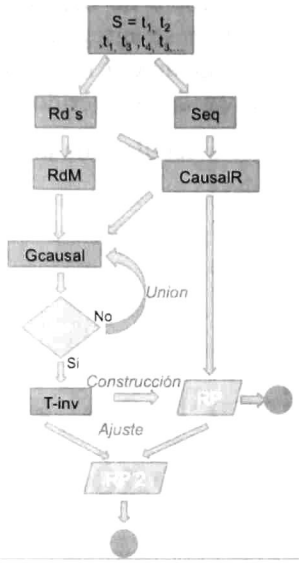


Figura 2.9. Diagrama de flujo minería de procesos vía Componentes Repetitivas

En la figura 2.9 se presenta el flujo de información desde la entrada al método hasta el modelo final de RP. Como entrada al método se da una secuencia de tareas *S* la cual puede ser de gran tamaño dependiendo la complejidad de los procesos. Con una simple observación de la secuencia *S* podemos obtener el conjunto *Seq* y los conjuntos de dependencias repetitivas entre las tareas *Rds*; después, con esta información detectamos las relaciones causales entre las tareas *CausalR*. Hasta aquí el método trabaja de forma similar al método propuesto en [Estrada, 2013].

Una vez que se tiene información sobre las dependencias entre las tareas podemos comenzar con el proceso de detectar los t-invariantes del modelo. Para esto primero extendemos los conjuntos *Rd* y los hacemos conjuntos maximales (*RdM*) los cuales contienen información más precisa acerca de las componentes repetitivas de nuestro modelo.

En el siguiente paso (rama izquierda del diagrama de flujo), los grafos de relaciones entre tareas son generados, uno por cada conjunto maximal del análisis anterior. El objetivo de estos grafos es formar componentes fuertemente conexas; en caso de no obtenerlas directamente en la construcción de los primeros grafos, se realizan uniones de los mismos; este proceso es iterativo y siempre finaliza al obtener la mayor cantidad de grafos fuertemente conexos que fue posible realizar a través de uniones.

Cuando estas iteraciones terminan los nodos de cada uno de lo grafos obtenidos forman los soportes de t-invariantes que debe tener nuestro modelo final; es momento de construir una primera aproximación del modelo, para esto se usan los t-invariantes y las relaciones

que fueron encontradas en el primer paso del método. Un conjunto de reglas de construcción simples producen una primera aproximación al modelo.

A continuación se comparan los t-invariantes del modelo obtenido con los obtenidos en el análisis; si estos resultan iguales nuestro modelo es correcto, en caso contrario será necesario realizar un ajuste al modelo. En la mayoría de los casos en que los t-invariantes no coinciden es resultado de la presencia de dependencias implícitas entre tareas en la secuencia *S*. Una vez realizado el ajuste se agregan lugares o arcos según sea el caso para restringir la ejecución del modelo resultante a la secuencia de transiciones *S*. Al final, el modelo ajustado admite sólo los t-invariantes inferidos.

En el siguiente capítulo se explica a detalle cada una de las partes de este método.

Capítulo 3

Síntesis de Redes de Petri Basada en la Inferencia de Componentes Repetitivas

Resumen. En este capítulo se presenta un nuevo método de minería de procesos de flujo de trabajo, el cual extiende el trabajo anterior incorporando al método la determinación de los t-invariantes y de las componentes repetitivas del modelo. Finalmente se obtiene un modelo de red de Petri a salvo que representa un registro de eventos observado en algún WMS.

3.1 Introducción

3.1.1 Motivación

La mayoría de los trabajos de minería de procesos de flujo de trabajo analizados en el capítulo 1, no hacen referencia a trabajos de identificación de eventos discretos, algunos de ellos revisados también en ese capítulo. Esto a pesar de que son problemas muy parecidos los cuales pueden atacarse con ambos métodos con algunas modificaciones; es por eso que se presenta en este trabajo un método de identificación para atacar el problema de minado de procesos de flujo de trabajo.

La segunda parte del método presentado en el capítulo 2 se acerca mucho a ser un problema de minería de procesos de flujo de trabajo, ya que la secuencia de eventos S producida por el sistema tiene suficiente información para poder generar, a partir de ella, un modelo que represente el mismo lenguaje que el observado en S. De aquí la idea de orientar este método haciendo algunas modificaciones al mismo, hacia minería de procesos.

En este trabajo se extienden los resultados obtenidos en [APE, 2013], se revisa la definición de dependencia repetitiva y se definen algunas otras propiedades que nos permiten a partir de la secuencia S obtener las componentes repetitivas del sistema. El hecho de conocer las componentes repetitivas, hace más simple el análisis de relaciones concurrentes y causales entre las tareas además de cambiar la manera en que la red final es construida.

3.1.2 Descripción general

En general el nuevo método de minado de procesos de flujo de trabajo presentado consta de 4 pasos los cuales se explican a continuación.

Paso 1. Detectar las dependencias repetitivas entre los eventos. En este paso se retoma y se extiende la definición de dependencia repetitiva con el fin de detectar eventos que pertenecen a esta dependencia. En este paso también se calculan conjuntos maximales que se acercan mucho a ser los soportes de los t-invariantes de la red.

- **Paso 2**. Determinación de las componentes repetitivas de la red. A partir de los conjuntos maximales se propone una técnica para verificar que efectivamente sean los soportes de t-invariantes de la red.

Paso 3 Construcción del modelo. con el conocimiento de las relaciones causales y concurrentes analizadas en el capítulo 2 y los t-invariantes de la red se expone una técnica para construir una primera aproximación de la red.

Paso 4 Ajuste final al modelo. Si los t-invariantes de modelo obtenido en el paso 3 no coinciden con los t-invariantes del análisis en el paso 2, es necesario realizar un ajuste al modelo. Si por el contrario coinciden estos, el modelo del paso 3 puede presentarse como resultado. En este paso se describe un método para ajustar el modelo y dar como resultado la red que representa exactamente el lenguaje en S.

3.2 Dependencia repetitiva entre las tareas

En el capítulo anterior fue presentada la definición de Dependencia Repetitiva, la cual dice que una transición t_i es dependiente repetitivamente de otra transición t_k, si y solo si t_k se observa en todos los casos entre dos apariciones de t_j. Dicho de otra forma, el conjunto de transiciones en dependencia repetitiva con t_i, $Rd(t_i)$, son aquellas transiciones que deben observarse siempre para poder ver la aparición de t_i en repetidas ocasiones.

Este razonamiento nos da una ligera sospecha de que este grupo de transiciones, son o forman parte de una componente repetitiva de nuestro modelo, sin embargo no tenemos el conocimiento suficiente, hasta el momento para poder decir que en realidad estas transiciones pertenecen a una verdadera componente repetitiva del sistema. Para asegurarnos de esto es necesario extender la definición de dependencia repetitiva. A continuación se mostrarán algunas definiciones y proposiciones necesarias para obtener los verdaderos soportes de t-invariantes del modelo a partir de los conjuntos de dependencias repetitivas.

Definición 3.1. Sea Y una solución entera positiva a la siguiente ecuación $C \cdot Y = 0$ entonces Y es llamado invariante de transiciones, t-invariante, o bien t-semiflujo y C es la matriz de incidencia de una RP

Definición 3.2. Un t-invariante es *mínimo (elemental o canónico),* cuando no puede expresarse como una combinación lineal de otros invariantes; y se expresa normalizado dividiéndolo por el máximo común divisor de los elementos del vector.

Definición 3.3. El soporte de un t-invariante es el conjunto de transiciones cuyos elementos correspondientes en un t-invariante son positivos. Un soporte mínimo corresponde a un invariante mínimo. Un t-invariante total (no necesariamente mínimo) tiene todos sus componentes positivos.

Proposición 3.1. Los soportes de T-invariantes mínimos de una RP la descomponen en componentes repetitivas (llamadas también T-componentes); el disparo de las transiciones de un soporte (no está definida la secuencia) a partir del marcado inicial, conduce a la red al mismo marcado. Si cada transición de una RP pertenece a algún soporte (existe un T-invariante total) la red es repetitiva.

Ejemplo: 3.1. La matriz de incidencia C para la red de Petri de la figura 3.1 a), tiene la siguiente forma cada fila representa un lugar y cada columna una transición en orden ascendente.

$$C = \begin{bmatrix} -1 & 1 & 0 & 0 \\ -1 & 1 & -1 & 1 \\ 0 & 0 & -1 & 1 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & -1 \end{bmatrix}$$

Para la matriz de incidencia C, se encontraron los siguientes soluciones enteras positivas $Y_1 = [1\ 1\ 0\ 0]^T$ y $Y_2 = [0\ 0\ 1\ 1]^T$ Por lo tanto son los t-invariantes de la red de Petri y sus soportes correspondientes son $Y_1 = [t_1, t_2]$ y $Y_2 = [t_3, t_4]$, la presentación grafica de estos t-invariantes es presentada en la figura 3.1 b).

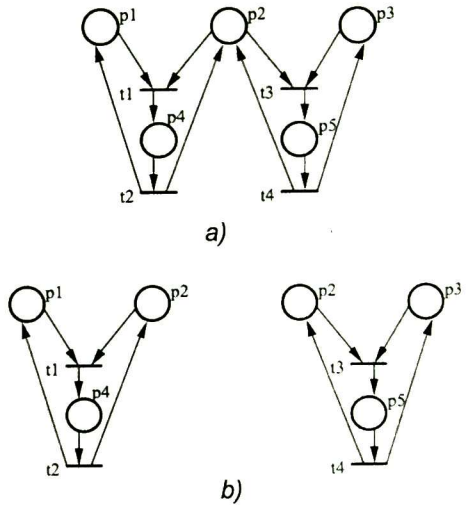


Figura 3.1. a) Red de Petri ej. 3.1 b) t-invariantes

3.2.1 Dependencia repetitiva Extendida

Basándonos en las definiciones y proposición antes mencionadas, podemos entonces proceder a explicar de forma detallada el proceso de cómo utilizar la información que se encuentra en los conjuntos de dependencias repetitivas Rd, para obtener los soportes de t-invariantes de nuestro sistema.

Proposición 3.2. Si a partir de la secuencia S se pueden determinar las relaciones $(t_c \prec t_b)$ and $(t_b \prec t_a)$, $t_c \in \mathrm{Rd}(t_b)$, $t_b \in \mathrm{Rd}(t_a)$, entonces deberá existir una transitividad entre la transición t_c y t_a.

Demostración: Si existen las relaciones $(t_c \prec t_b)$ y $(t_b \prec t_a)$ en S, pero $t_c \notin Rd(t_a)$ entonces nuestro sistema podría generar secuencias del siguiente tipo, $t_a\ t_x\ t_b\ t_x\ t_a$ repetidamente, donde t_x puede ser cualquier transición del sistema pero no t_c dado que como hemos dicho antes esta relación no existe, al ejecutar repetidamente esta secuencia generaremos secuencias como ésta $t_a\ t_x\ t_b\ t_x\ t_a\ t_x\ t_b\ t_x\ t_{a\ldots}$ pero esta secuencia no tiene la relación $(t_c \prec t_b)$ ya que podemos ejecutar repetidamente t_b sin necesidad de la aparición de t_c , sin embargo esta es una contradicción por que la relación $(t_c \prec t_b)$ existe en nuestro análisis. Por lo tanto si queremos observar la transición t_a más de dos veces será necesaria la presencia de t_c tal y como se muestra en la siguiente secuencia $t_a\ t_x\ t_b\ t_x\ t_{a\ldots}\ t_{c\ldots}\ t_x\ t_b\ t_x\ t_{a\ldots\ldots}$ y por lo tanto t_c deberá estar incluida en el conjunto de dependencias repetitivas de t_a.

Ejemplo 3.2 De la siguiente secuencia $S = t_2\ t_4\ t_1\ t_4\ t_2\ t_1\ t_4\ t_2\ t_1\ t_3\ t_2\ t_1\ t_2\ t_3\ t_1\ t_2\ t_4\ t_1\ t_3\ t_2\ t_1\ t_4\ t_2\ t_1\ t_3\ t_2\ t_1\ t_2\ t_3\ t_1\ t_3\ t_2\ t_1\ t_3\ t_2\ t_1\ t_4\ t_2\ t_1\ t_3\ t_2\ t_1\ t_2\ t_3\ t_1\ t_2\ t_4\ t_1\ t_3\ t_2\ t_1\ t_4\ t_2\ t_1$. Se obtienen los siguientes conjuntos de dependencias repetitivas, $Rd(t_1) = \{ t_1, t_2 \}$, $Rd(t_2) = \{ t_1, t_2 \}$, $Rd(t_3) = \{ t_1, t_3 \}$, $Rd(t_4) = \{ t_1, t_4 \}$. Donde se tienen las relaciones $(t_2 \prec t_1)$ y $(t_1 \prec t_4)$, al igual que en la Demostración anterior tomemos como $t_a t_b t_c$ como $t_2\ t_1\ t_4$ respectivamente tomamos de S la única secuencia que nos permite observar más de dos veces a t_4, "$t_4\ t_1\ t_4\ t_2\ t_1\ t_4$", esto es, no podemos observar repetidamente la transición t_4 sin la presencia de la transición t_2.

Definición 3.4. Una dependencia repetitiva indirecta entre dos transiciones t_c , t_a denotada como $(t_c \prec\prec t_a)$, existe ssi la secuencia S presenta las relaciones $(t_c \prec t_b)$, $(t_b \prec t_a)$, el conjunto de dependencias repetitivas indirectas para una transición es definido como $IRd(t_a)=\{t_c \mid t_c \prec\prec t_a\}$.

Definición 3.5. El conjunto de dependencias repetitivas extendido de una transición t_a, es aquel conjunto que contiene, tanto las transiciones incluidas en su $Rd(t_a)$, como aquellas incluidas después de aplicar la extensión transitiva entre los conjuntos de dependencias repetitivas $IRd(t_a)$:

$$Rd_{ex}(t_a) = \{Rd(t_a) \cup IRd(t_a)\}$$

Ejemplo 3.3 Para el ejemplo 3.2 los conjuntos de dependencias repetitivas extendidos para cada una de las transiciones son los siguientes, $Rd_{ex}(t_1) = \{ t_1, t_2 \}$, $Rd_{ex}(t_2) = \{ t_1, t_2 \}$, $Rd_{ex}(t_3) = \{ t_1, t_3, t_2\}$, $Rd_{ex}(t_4) = \{ t_1, t_4, t_2\}$. Tanto para el $Rd_{ex}(t_3)$ y el $Rd_{ex}(t_4)$ la transición t_2 se ha agregado ya que existen las relaciones $(t_2 \prec t_1)$, $(t_1 \prec t_4)$ $(t_2 \prec\prec t_4)$, $(t_2 \prec t_1)$, $(t_1 \prec t_3)$ $(t_2 \prec\prec t_3)$. La figura 3.2 muestra estas relaciones de forma gráfica en un grafo de relaciones de dependencia repetitiva entre las tareas, las aristas con línea punteada indican las relaciones agregadas después de la extensión transitiva.

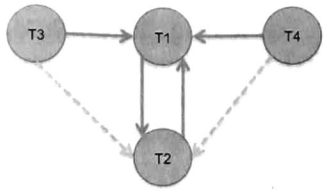


Figura 3.1 grafo de relaciones de *Rd* Y *IRd*

3.2.2 Dependencia repetitiva maximal

Hablamos antes de la relación que existe entre la dependencia repetitiva de una transición obtenida de la observación de la secuencia *S* y los t-invariantes del proceso real de donde fue obtenida la secuencia. Sin embargo es necesario idear un procedimiento para juntar todos aquellos conjuntos de dependencias repetitivas que pertenezcan a una misma componente repetitiva del sistema, ya que cada *Rd* contienen transiciones que forman parte de una de estas componentes repetitivas. En ocasiones pueden ser completas pero para la mayoría de secuencias *S*, será necesario juntar estos conjuntos para hacer conjuntos más grandes hasta obtener dichas componentes repetitivas.

Antes de entrar de lleno a este proceso analicemos algunas propiedades que pueden ser inferidas a partir de la secuencia *S* y el análisis de los conjuntos de dependencias repetitivas indirectas para cada transición (*IRd*). Calculamos los *IRds* de la secuencia del ejemplo 3.2 los cuales fueron $Rd_{ex}(t_1) = \{ t_1, t_2 \}$, $Rd_{ex}(t_2) = \{ t_1, t_2 \}$, $Rd_{ex}(t_3) = \{ t_1, t_3, t_2\}$, $Rd_{ex}(t_4) = \{ t_1, t_4, t_2\}$, todos estos conjuntos tienen en común las transiciones t_1, t_2, esto significa que siempre estarán presentes en una misma componente repetitiva del sistema.

Proposición 3.3. Si en los conjuntos de dependencias repetitivas extendidas (IR_{ex}) encontramos un grupo de transiciones comunes en todos los IR_{ex}, estas transiciones deberán estar presentes en cada componente repetitiva que tenga nuestro sistema.

Demostración: Partiendo de la definición *2.5* de dependencia repetitiva entre transiciones, la cual hereda sus propiedades a IR_{ex}, ésta dice que un $Rd(t_j)$ contiene las transiciones que deben ser ejecutadas para rehabilitar el disparo de t_j. Sea $A \subseteq T$ un grupo de transiciones que están en cada uno de los *IRd* y Sea $t_x \in A$ cualesquiera de estas transiciones. Para poder disparar repetitivamente t_x es necesaria la presencia de todas las transiciones en A, como cada IR_{ex} contienen todas y cada una de las transiciones t_x, el grupo de transiciones A, estará presente en cada ciclo repetitivo de la red y por lo tanto también en cada componente repetitiva de la misma.

Haciendo uso de la proposición 3.2 nos podemos definir un conjunto maximal de Dependencias repetitivas que se acerque más a ser una componente repetitiva de nuestro sistema.

Definición 3.6 Un conjunto de Dependencia repetitiva maximal denotado como RdM_x, es un $IRd_{ex}(t_i)$ el cual no puede ser incluido en ningún otro $IRd_{ex}(t_j)$ y se define con la siguiente expresión.:

$$\mathrm{RdM_x} = \{\, \mathrm{Rd_{ex}(t_i)} \mid \nexists \mathrm{Rd_{ex}(t_j)} \subset \mathrm{Rd_{ex}(t_i)} \}.$$

Ejemplo 3.4 Los conjuntos de dependencias repetitivas maximales para los $Rd_{ex}s$ calculados en el ejemplo 3.2, son los siguientes, $RdM_1 = \{ t_1, t_3, t_2 \}$, $RdM_2 = \{ t_1, t_4, t_2 \}$. Ya que ambos $Rd_{ex}(t_3)$ y $Rd_{ex}(t_4)$ respectivamente no pueden ser incluidos por ningún otro Rd_{ex} ni tampoco entre ellos mismos. Por otro lado los conjuntos $Rd_{ex}(t_2)$ y $Rd_{ex}(t_1)$ son incluidos en estos *RdM* mencionados anteriormente, la figura 3.2 muestra gráficamente este razonamiento.

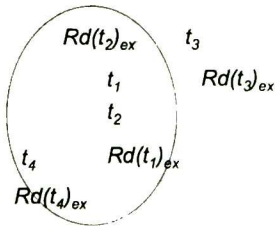


Figura 3.2 *RdMs* de ejemplo 3.3

Proposición 3.4. Sea t_x una transición que está presente solo en una de las dependencias repetitivas máximales del sistema *RdM* y t_y otra transición con la misma característica. Si t_x, t_y son observadas en el conjunto de relaciones concurrentes, $(t_x, t_y) \in ConcR$ y sus conjuntos de dependencias repetitivas maximales correspondientes RdM_x, RdM_y respectivamente y también sus dependencias repetitivas simples $Rd(t_x)$, $Rd(t_y)$ comparten transiciones en común, entonces t_x, t_y deben pertenecer a una misma dependencia repetitiva maximal.

Demostración: Para que dos transiciones t_x, t_y observadas siempre en diferente dependencia repetitiva maximal puedan tener un comportamiento concurrente, es necesario que los conjuntos maximales que contienen a éstas transiciones RdM_x, RdM_y, sean disjuntos, ya que estos conjuntos de transiciones podrían ejecutarse repetitivamente de forma independiente uno del otro y en algún momento t_x, t_y pueden observarse concurrentes en esta ejecución paralela. En caso de que la intersección de RdM_x, RdM_y, sea diferente de

cero, entonces la intersección de sus conjuntos de dependencias repetitivas simples de estos eventos $Rd(t_x)$, $Rd(t_y)$ necesitara ser vacía para así asegurar que por lo menos uno de estos pueda ejecutarse repetidamente y provocar en algún momento de la ejecución la concurrencia de t_x y t_y. Por el contrario supongamos que $Rd(t_x) \cap Rd(t_y) \neq \emptyset$ y sea $t_z \in Rd(t_x) \cap Rd(t_y)$, cualquier elemento de la intersección, entonces por definición *2.5* t_x, t_y necesitan de t_z para ejecutarse repetidamente además $(t_x, t_y) \in ConcR$, por lo tanto existe un comportamiento repetitivo en la secuencia S que genera las siguiente cadena $A \ldots t_z \ldots A \ldots t_z A$ donde A pueden tomar el valor de $t_x\ t_y$ o bien $t_y\ t_z$ y esto solo es posible si t_x, t_y pertenecen a la misma componente repetitiva.▪

El análisis anterior se muestra de manera gráfica en la figura 3.3. En la figura 3.3 a) se muestra el caso en que $RdM_x \cap RdM_y = \emptyset$, es posible ver t_x, t_y concurrentes porque ambos ciclos se ejecutan independientemente; b) muestra el caso en que $RdM_x \cap RdM_y \neq \emptyset$ pero $Rd(t_x) \cap Rd(t_y) = \emptyset$, uno de los ciclos puede ejecutarse repetitivamente sin necesidad de alguna transición del otro y en algún momento observarse t_x, t_y concurrentes; en c) $RdM_x \cap RdM_y \neq \emptyset$ y $Rd(t_x) \cap Rd(t_y) \neq \emptyset$, por lo tanto como se puede observar en la figura $t_x\ t_y\ t_z$ están dentro de una misma componente repetitiva.

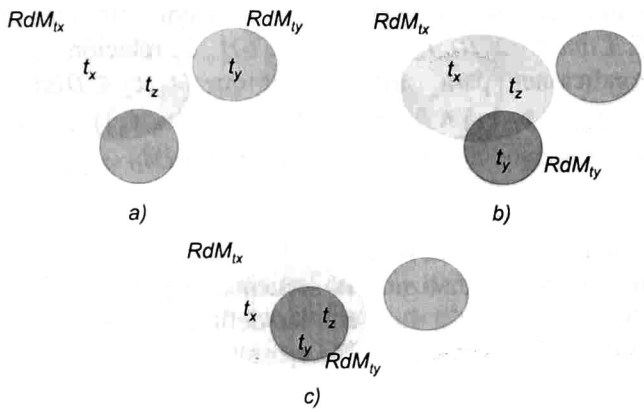


Figura 3.3. Explicación gráfica de la demostración a la proposición 3.4

Definición 3.7. El conjunto de transiciones que aparecen en solo uno de los *RdMs* del sistema, denotado como *Dist,* se define de la siguiente manera.

$$Dist = \bigcup_{i=1}^{n} \left(RdM_i \setminus \bigcup_{j=1; j \neq i}^{n} RdM_j \right)$$

Ejemplo 3.5. El conjunto *Dist* para los *RdMs* encontrados en el ejemplo 3.4 $RdM_1 = \{ t_1, t_3, t_2 \}$, $RdM_2 = \{ t_1, t_4, t_2 \}$, se calcula de la siguiente manera $Dist = (RdM_1 \setminus RdM_2) \cup (RdM_2 \setminus RdM_1) = \{t_3, t_4\}$.

Definición 3.8. $RdM_{t_x,t_y} = RdM_{t_x} \cup RdM_{t_y}$ si y solo si $(t_x, t_y \in Dist) \wedge ((t_x, t_y) \in ConcR) \wedge (RdM_{t_x} \cap RdM_{t_y} \neq \emptyset) \wedge (Rd(t_x) \cap Rd(t_y) \neq \emptyset)$ donde RdM_{t_x} RdM_{t_y} son los RdMs que contienen a la transición t_x, t_y respectivamente. Si existe una relación del

tipo RdM_{t_x,t_y} se pone ésta en el grupo de *RdMs* y tanto RdM_{t_x} como RdM_{t_y} son eliminados de los *RdMs.*

Ejemplo 3.6. Para la secuencia del ejemplo 3.2 la cual ha sido analizada desde el inicio del capítulo no existe ninguna relación de la forma RdM_{t_x,t_y} ya que nuestro conjunto de concurrencias es $ConcR = \{(t_2, t_3), (t_2, t_4)\}$, y ninguno de estos pares de transiciones están en el conjunto $Dist = \{t_3, t_4\}$ antes calculado y basta con que alguna de las condiciones en la fórmula no se cumpla, ya que es una fórmula conjuntiva. Entonces nuestros conjuntos *RdM* permanecen sin cambio.

En el siguiente ejemplo trataremos una cadena que exhibe la relación RdM_{t_x,t_y} con el objetivo de dejar clara esta definición.

Ejemplo 3.7. Para la siguiente cadena $S = t_0\ t_5\ t_2\ t_1\ t_6\ t_4\ t_0\ t_5\ t_2\ t_1\ t_6\ t_4\ t_0\ t_5\ t_2\ t_1\ t_6\ t_4\ t_0\ t_2\ t_5\ t_1\ t_6\ t_4\ t_0\ t_2\ t_5\ t_1\ t_6\ t_4\ t_0\ t_5\ t_3\ t_6\ t_4\ t_0\ t_2\ t_5\ t_1\ t_6\ t_4\ t_0\ t_5\ t_2\ t_1\ t_6\ t_4\ t_0\ t_5\ t_1\ t_6\ t_2\ t_4\ t_0\ t_2\ t_5\ t_1\ t_6\ t_4\ t_0\ t_2\ t_5\ t_1\ t_6\ t_4\ t_0\ t_2\ t_5\ t_1\ t_6\ t_4\ t_0\ t_5\ t_3\ t_6\ t_4\ t_0\ t_2\ t_5\ t_1\ t_6\ t_4\ t_0\ t_5\ t_2\ t_1\ t_6\ t_4\ t_0\ t_2\ t_5\ t_1\ t_6\ t_4\ t_0\ t_2\ t_5\ t_1 \ldots$. Los conjuntos de dependencias repetitivas encontrados para esta secuencia son los siguientes, $Rd(t_0) = \{ t_0, t_5, t_6, t_4 \}$, $Rd(t_1) = \{ t_1, t_0, t_5, t_6, t_4 \}$, $Rd(t_2) = \{ t_2, t_0, t_4\}$, $Rd(t_3) = \{ t_3, t_0, t_5, t_6, t_4 \}$, $Rd(t_4) = \{ t_0, t_5, t_6, t_4 \}$, $Rd(t_1) = \{ t_5, t_0, t_5, t_6 \}$ y $Rd(t_6) = \{t_0, t_5, t_6, t_4 \}$, después de haber calculado la dependencia repetitiva extendida Rd_{ex} para cada uno de los conjuntos, llegamos a tener los siguientes *RdMs*, $RdM_1 = \{ t_2, t_0, t_5, t_6, t_4\}$, $RdM_2 = \{ t_1, t_0, t_5, t_6, t_4\}$ y $RdM_3 = \{ t_3, t_0, t_5, t_6, t_4\}$, de estos calculamos el conjunto $Dist = \{ t_1, t_2, t_3\}$ y por ultimo las relaciones observadas concurrentes en *S*, $ConcR = \{ (t_1, t_2), (t_2, t_5), (t_2, t_3)\}$ la relación $(t_1, t_2) \in ConcR$ cumple con todas las condiciones para RdM_{t_1,t_2} ya que $(t_1, t_2 \in Dist \wedge (t_1, t_2) \in ConcR \wedge RdM_{t_1} \cap RdM_{t_2} = \{t_0, t_5, t_6, t_4\} \wedge Rd(t_1) \cap Rd(t_2) = \{t_0, t_4\})$ entonces $RdM_{t_1,t_2} = RdM_{t_1} \cup RdM_{t_2}$ y los nuevos *RdMs* del sistema serán $RdM_{t1,t2} = \{ t_2, t_2, t_0, t_5, t_6, t_4\}$, $RdM_3 = \{ t_3, t_0, t_5, t_6, t_4\}$.

3.3 Soportes de t-invariantes de la secuencia *S*

Hasta este momento se ha analizado la secuencia *S* para determinar las componentes repetitivas del sistema. Sin lugar a duda con las definiciones anteriores es posible obtener esta información solo de la secuencia *S.* Por ejemplo el grupo de dependencias repetitivas maximales *RdMs* son o forman parte de alguno de los soportes de t-invariantes del sistema, pero es necesario asegurarnos que están completos, para poder decir entonces que son los verdaderas componentes repetitivas. Para esto es necesario introducir nuevos conceptos y proposiciones que nos ayudarán a lograr tal objetivo.

3.3.1 Grafo de *RdM G.*

Un grafo de un conjunto de dependencia repetitiva maximal, es un grafo dirigido que tiene como nodos, todas y cada una de las transiciones que están contenidas en él y como aristas tiene las relaciones entre las tareas que fueron observadas en *S,* tanto las causales como aquellas que - fue imposible detectar como concurrentes o bien como causales. Antes de pasar a la definición formal del grafo de *RdM G,* veamos algunas definiciones que le conciernen.

Definición 3.9. El conjunto *R* es un conjunto que contiene todas a aquellas relaciones que fueron observadas causales $(t_a, t_b) \in CausalR$, más aquellas transiciones en $Seq' = (Seq \setminus CausalR) \setminus ConcR) \setminus ConcR^{-1}$, es decir que fueron observadas consecutivas alguna

vez en S ($t_a < t_b$), pero no están dentro de una relación concurrente o causal; donde $ConcR^{-1} = \{(t_b, t_a) | (t_a, t_b) \in ConcR\}$. El conjunto R es definido de la siguiente manera.

$$R = \{(t_a, t_b) | (t_a, t_b) \in CausalR \vee (t_a, t_b) \in Seq\}$$

Definición 3.20. El grafo de relaciones entre de las tareas de un RdM_i denotado como G_i, es definido formalmente como.

$$G_i = (V_i, E_i)$$

$$V_i = \{t_a | t_a \in RdM_i\}$$

$$E_i = \{(t_a, t_b) \in V \times V | (t_a, t_b) \in R\}$$

Ejemplo 3.8. Analizando la secuencia S del ejemplo 3.2 podemos obtener los siguientes conjuntos de relaciones $Seq = \{ (t_2 < t_1), (t_2 < t_4), (t_2 < t_3), (t_4 < t_1), (t_4 < t_2), (t_1 < t_2), (t_1 < t_3), (t_1 < t_4) , (t_3 < t_1), (t_3 < t_2)\}$, $Tc = \{ (t_1 , t_2), (t_1 , t_3) \}$, $ConcR = \{ (t_2, t_4), (t_2, t_3) \}$ $CausalR = \{ (t_2 , t_1), (t_4 , t_1), (t_1 , t_2), (t_1 , t_3), (t_1 , t_4) , (t_3 , t_1) \}$, $Seq = \{\emptyset\}$. La relación $R = CausalR$ ya que en este caso nuestra secuencia S es completa , los siguientes *RdMs* calculados en el ejemplo 3.4 para esta misma secuencia son $RdM_1 = \{ t_1, t_3, t_2\}$, $RdM_2 = \{ t_1, t_4, t_2\}$. La figura 3.4 muestra los grafos G de estas dependencias repetitivas maximales.

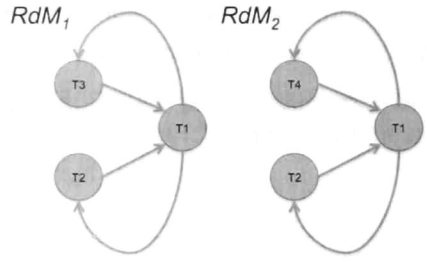


Figura 3.4 Grafos *G*, para los *RdMs* del ejemplo 3.3

Definición 3.21. Un grafo que tiene un camino desde cualquiera de sus vértices a cualquiera de sus otros vértices se dice que es *Fuertemente Conexo* denotado como ***Sc***, y su definición formal se presenta a continuación. Sea $G = (V, E)$ un grafo dirigido, tal que para todo par de vértices $u, v \in V$, existe un camino en G que va de u hasta v, y otro de v a u.

El objetivo de realizar los grafos es verificar si obtenemos como resultado un grafo fuertemente conexo, tal es el caso de los grafos en la figura 3.4, esta es una propiedad interesante porque nos acerca a conocer los verdaderos soportes de las componentes repetitivas de nuestra net(t-invariantes). Desafortunadamente, nuestro procedimiento para obtener los grafos no siempre produce grafos que no son *Sc*; este resultado se presenta debido a que los conjuntos de dependencias repetitivas maximales no son completos, es decir hace falta que contengan algunas transiciones que nos ayuden a completar los ciclos en el grafo y así formar grafos fuertemente conexos.

A continuación se presenta una técnica para saber exactamente cuáles son estas transiciones faltantes y obtener los soportes de t-invariantes de nuestro modelo.

3.3.2 Uniones entre Grafos de *RdM G*

Como se dijo antes el objetivo de estas uniones entre grafos, es obtener grafos fuertemente conexos (*Sc*), a continuación se presenta la definición de unión entre grafos.

Definición 3.22 La unión entre dos grafos de dependencias repetitivas maximales G_1 . G_2 los cuales comparten nodos entre sí, da como resultado un nuevo grafo $G_{1,2}$ el cual se define de la siguiente manera.

$$G_{1,2} = (V_{1,2}, E)$$

$$V_{1,2} = \{t_a | t_a \in (V_1 \cup V_2)\}$$

$$E = \{(t_a, t_b) \in V_{1,2} \times V_{1,2} | (t_a, t_b) \in R\}$$

Donde V_1, V_2 son los vértices de los grafos G_1, G_2 respectivamente.

Proposición 3.5. Si el grafo de relaciones de dependencia repetitiva maximal G o resultado de la unión de grafos $G_{x,y}$, Es un grafo Sc, entonces los nodos que conforman dicho grafo son el soporte de algún t-invariante del sistema.

Demostración: Basados en la definición 3.2, sabemos que los soportes de T-invariantes mínimos de una RP descomponen a esta en componentes repetitivas, donde el disparo de sus transiciones a partir del marcado inicial, conduce a la red al mismo marcado. Podemos ver entonces el grafo G como una de estas componentes repetitivas inducidas por t-invariantes, en la cual debemos de ser capaces de realizar un camino en G, partiendo de cualquiera de sus nodos y concluir en este mismo, lo que representa una condición necesaria pata ejecutar las transiciones del soporte de un t-invariante.

El siguiente procedimiento permite encontrar los soportes de t-invariantes del sistema; el algoritmo recibe como entrada el conjunto $G = \{G_1, G_2, G_3, ..., G_n\}$ donde G_i es el grafo de relaciones de RdM_i.

Algoritmo 3.1. Determinación de los t-invariantes

Entrada: G
Salida: G^{Sc}: Grafos Sc

1. *New-Sc* ← verdadero; *Aux* ← ∅; *Nsc* ← $G_j \in G \mid G_j$ no *Sc*
2. Mientras *New-Sc* = verdadero
   2.1 Para todo $G_i \in Nsc$
      Para todo $G_j \in G \mid G_j \neq G_i$ y $V_i \cap V_j \neq \emptyset$
      $G_{i,j} = G_i \cup G_j$
      Si $G_{i,j}$ es *Sc*
         Entonces $G^{Sc} = G^{Sc} + \{ G_{i,j} \}$, *ban = verdadero*
         De lo contrario
         Si $G_{i,j} \not\subseteq Aux$
         Entonces Aux = Aux + $G_{i,j}$; *New-Sc* ← falso
   2.2 *Nsc* ← *Aux*; *Aux* ← ∅

Los vértices de cada grafo $G_i \in G^{Sc}$ constituyen cada uno de los soportes de t-invariantes de la red.

Presentaremos un ejemplo en el cual, es necesario el uso de uniones entre grafos de dependencias repetitivas máximas, para explicar con detalle el procedimiento anterior.

Ejemplo 3.9. Para la siguiente secuencia $S = t_4\ t_2\ t_3\ t_5\ t_4\ t_0\ t_3\ t_1\ t_6\ t_4\ t_0\ t_3\ t_1\ t_5\ t_4\ t_2\ t_3\ t_5\ t_4\ t_0\ t_1\ t_3\ t_5\ t_4\ t_2\ t_3\ t_6\ t_4\ t_0\ t_3\ t_1\ t_6\ t_4\ t_2\ t_3\ t_6\ t_4\ t_2\ t_3\ t_5\ t_4\ t_0\ t_1\ t_3\ t_5\ t_4\ t_3\ t_2\ t_5\ t_4\ t_3\ t_0\ t_1\ t_6\ t_4\ t_2\ t_3\ t_5\ t_4\ t_3\ t_0\ t_1\ t_5\ t_4\ t_3\ t_0\ t_1\ t_5\ t_4\ t_2\ t_3\ t_5\ t_4\ t_2\ t_3\ t_6\ t_4\ t_3\ t_2\ t_6\ t_4\ t_3\ t_0\ t_1\ t_5\ t_4\ t_3\ t_0\ t_1\ t_5\ t_4\ t_3\ t_0\ t_1\ t_5\ t_4\ t_3\ t_0\ t_1\ t_6\ t_4\ t_0\ t_1\ t_3\ t_5\ t_4\ t_3\ t_2\ t_6\ t_4\ t_0\ t_1\ t_3\ t_6\ t_4\ t_0\ t_1\ t_3\ t_6\ t_4\ t_0\ t_3\ t_1 \ldots$ se obtuvieron las siguientes relaciones entre transiciones:

- $Seq = \{ (t_0 < t_3), (t_0 < t_1), (t_1 < t_3), (t_1 < t_5), (t_1 < t_6), (t_2 < t_3), (t_2 < t_5), (t_2 < t_6), (t_3 < t_2), (t_3 < t_5), (t_3 < t_6), (t_3 < t_1), (t_3 < t_6), (t_4 < t_2), (t_4 < t_3), (t_4 < t_0), (t_5 < t_4), (t_6 < t_4) \}$, $Tc = \emptyset$, $ConcR = \{ (t_0, t_3), (t_1, t_3), (t_2, t_3) \}$
- $CausalR = \{ (t_0, t_1), (t_3, t_5), (t_3, t_6), (t_4, t_2), (t_4, t_3), (t_4, t_0), (t_5, t_4), (t_6, t_4) \}$, $Seq' = \{(t_1 < t_5), (t_1 < t_6), (t_2 < t_5), (t_2 < t_6)\}$.
- $R = CausalR \cup Seq' = \{ (t_0, t_1), (t_3, t_5), (t_3, t_6), (t_4, t_2), (t_4, t_3), (t_4, t_0), (t_5, t_4), (t_6, t_4), (t_1, t_5), (t_1, t_6), (t_2, t_5), (t_2, t_6) \}$.
- Los conjuntos de dependencias repetitivas máximas son $RdM_0 = \{ t_4, t_0, t_1, t_3 \}$, $RdM_1 = \{ t_2, t_4, t_3 \}$, $RdM_2 = \{ t_5, t_4, t_3 \}$, $RdM_3 = \{ t_6, t_4, t_3 \}$,
- El conjunto de transiciones que aparecen en uno solo de los *RdMs* es $Dist = \{ t_0, t_1, t_2, t_5, t_6 \}$.

Para la secuencia *S* de este ejemplo no es necesario modificar los *RdMs* antes de realizar los grafos ya que no existen transiciones t_a, t_b tales que $(t_a, t_b) \in ConcR$ y $t_a, t_b \in Dist$, siguiendo el procedimiento descrito anteriormente. El conjunto *G* de entrada al algoritmo consta de los grafos mostrados en la figura 3.5.

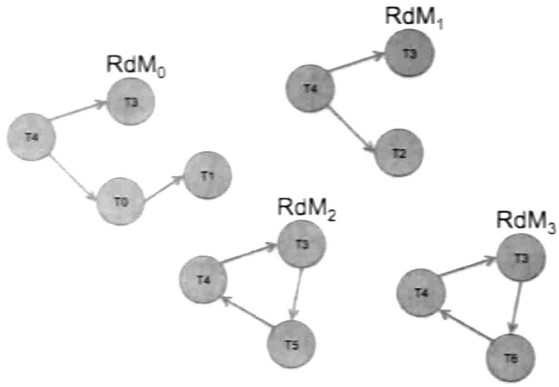


Figura 3.5 Grafos de *RdMs* para el ejemplo 3.9.

Podemos observar que los grafos RdM_0 y RdM_1 no son *Sc*, por lo tanto es necesario realizar la unión de grafos.

En la figura 3.6.a se muestra la unión de RdM_0 con el RdM_1 la cual da como resultado un nuevo grafo $RdM_{0.1}$, el cual no es fuertemente conexo; por otro lado, la figura 3.6.b muestra la unión de RdM_0 con RdM_2, en este caso el grafo resultante $RdM_{0.2}$ es un grafo *Sc*.

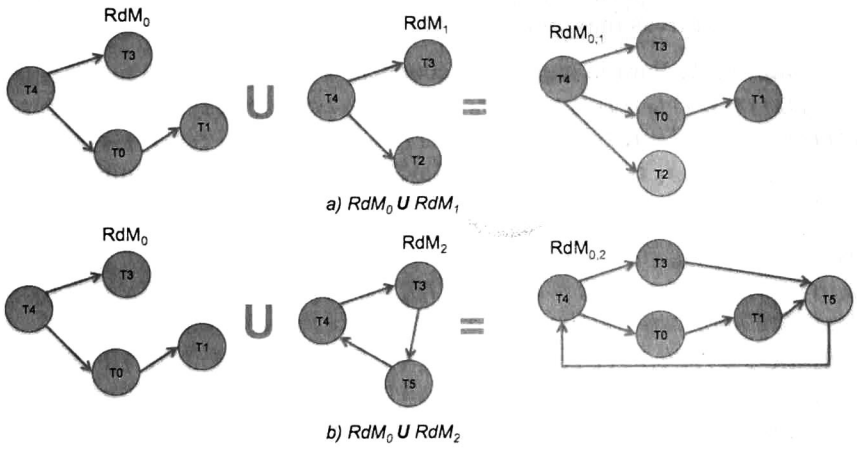


Figura 3.6. Uniones entre grafos de *RdMs*.

Después de que el algoritmo termina, obtenemos los siguientes grafos *Sc* de la figura 3.7 cuyo nombre hace referencia (en el subíndice) a *RdMs* de las cuales son resultado.

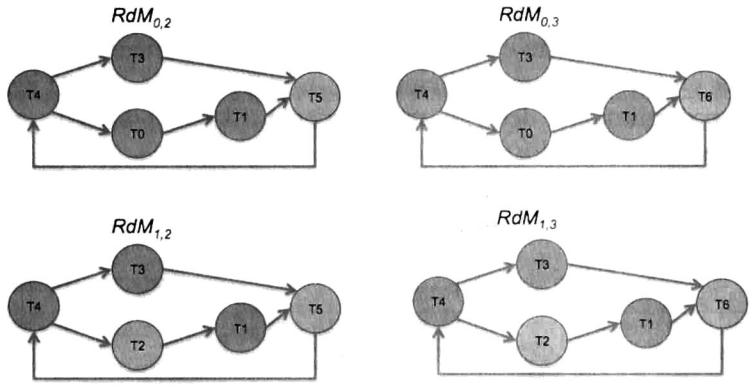


Figura 3.7 Grafos de *RdMs Sc* para el ejemplo 3.9.

Una vez concluido el proceso podemos decir que los soportes de t-invariantes para el sistema de la secuencia *S*, son exactamente los nodos que pertenecen a los grafos de la figura 3.7, $<Y_0>=\{t_0, t_4, t_1, t_3, t_5\}$, $<Y_1>=\{t_0, t_4, t_1, t_3, t_6\}$, $<Y_2>=\{t_2, t_4, t_3, t_5\}$, $<Y_3>=\{t_2, t_4, t_3, t_6\}$.

3.3 Construcción del modelo

Con la información de las relaciones entre las transiciones y conociendo los soportes de los t-invariantes, podemos entonces proponer una técnica para construir el modelo (red de Petri) que sea capaz de reproducir la secuencia *S*.

Proposición 3.6. Si para dos transiciones t_a, t_b existe en *S* la relación$(t_a \prec t_b)$, $t_a \in Rd(t_b)$ entonces, debe existir en el modelo (RP_1), un circuito simple y elemental (circuito *SE*) al cual ambas transiciones pertenecen [Estrada, 2013].

Demostración. Supongamos que no existe un circuito que contenga las transiciones t_a, t_b entonces justo después del disparo de t_b todos los tokens en t_b podrían desplazarse a

través de las transiciones de algún camino hasta · t_b, lo cual habilitaría a t_b sin la necesidad de ejecutar t_a, pero esto implica una contradicción que $t_a \notin Rd(t_b)$.

Proposición 3.7. Si para dos transiciones t_a, t_b existen en S las relaciones $(t_a < t_b)$ y $(t_a \prec t_b)$, $t_a \in Rd(t_b)$ ó $(t_b \prec t_a)$, $t_b \in Rd(t_a)$ entonces en el modelo (RP_1) debe existir un lugar que conecte t_a con t_b.

Demostración. Supóngase que no existe la presencia de un lugar entre t_a, t_b, entonces para poder observar la relación $(t_a < t_b)$ t_a, t_b deberían ser ejecutadas simultáneamente, ayudándonos de la proposición 3.6, si observamos t_a, t_b habilitadas simultáneamente y ejecutamos t_a todos los caminos de t_a hacia t_b tienen dos tokens, si todas las transiciones de un camino de t_a hacia t_b son ejecutadas, tendríamos entonces dos tokens en alguno de los lugares de entrada de t_b tendríamos una red de Petri que no es segura (1-acotada y viva). Ahora bien por lo menos una de las transiciones en cada camino de t_a *a* t_b estará condicionada al previo disparo de t_b. Pero si t_b es ejecutada, todas las transiciones en los caminos de t_a *a* t_b podrían ser ejecutados y tendríamos caminos de t_b *a* t_b que no necesitarían de la observación de la transición t_a, y como consecuencia $t_a \notin Rd(t_b)$. De igual manera para la relación $t_b \in Rd(t_a)$.

Proposición 3.8. Si $(t_a, t_b) \in Tc$, Entonces debe existir un lugar de t_a *a* t_b y otro de t_b *a* t_a.

Demostración. La secuencia t_a t_b t_a debe ser reproducida en RP_1, entonces justo después de la ejecución de t_a existe un token en sus lugares de salida y t_b estará a la salida de alguno de estos lugares; de lo contrario habrá dos tokens en estos lugares después del segundo disparo de t_a. De manera similar, justo después del primer disparo de t_a, no habrá tokens en sus lugares de entrada, y entonces t_b debe estar en la entrada de estos lugares; de lo contrario, t_a no podría ejecutarse de nuevo, de igual forma para la secuencia t_b t_a t_b.

3.3.1 Reglas de construcción

Para el conjunto de transiciones en la relación R, si $(t_a, t_b) \in CausalR$ debe existir un lugar entre este par de transiciones y será denotado por $[t_a, t_b]$ tal y como se muestra en la siguiente figura.

Figura 3.9. Lugar de t_a a t_b

Proposición 3.9. Si encontramos en R relaciones que comparten la misma transición de salida o bien la misma transición de entrada, es decir $[t_b, t_a]$, $[t_c, t_a]$ *o* $[t_a, t_b]$, $[t_a, t_c]$ y además las transiciones t_b, t_c pertenecen a un mismo t-invariante. Se aplican las siguientes reglas de construcción.

- $[t_a, t_b]$ $[t_a, t_c] \rightarrow [t_a, (t_b \,||\, t_c)]$

  $[t_b, t_a]$ $[t_c, t_a] \rightarrow [(t_b \,||\, t_c), t_a]$

Proposición 3.10. Si encontramos en R relaciones que comparten la misma transición de salida o bien la misma transición de entrada, es decir $[t_b, t_a]$, $[t_c, t_a]$ *o* $[t_a, t_b]$, $[t_a, t_c]$ y

además las transiciones t_b, t_c no pertenecen a un mismo t-invariante. Se aplican las siguientes reglas de construcción.

$[t_a, t_b]\ [t_a, t_c] \rightarrow [t_a, (t_b + t_c)]$ Siempre y cuando $(t_b , t_c) \notin ConcR$

$[t_b, t_a]\ [t_c, t_a] \rightarrow [(t_b + t_c), t_a]$ Siempre y cuando $(t_b , t_c) \notin ConcR$

Estas reglas de construcción pueden ser generalizadas para n transiciones de entrada o bien n transiciones de salida tal y como se muestra en la figura 3.10.

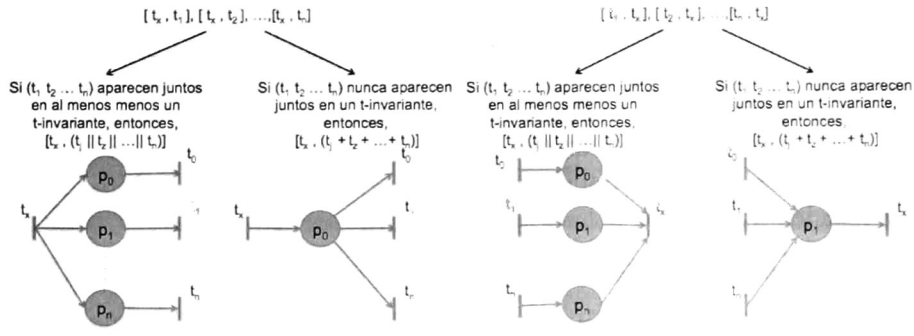


Figura 3.10 Representación grafica de las reglas de construcción del modelo

Ejemplo 3.10. Para la información obtenida de la secuencia *S* del ejemplo anterior (ejemplo 3.9), podemos obtener el siguiente conjunto de relaciones aplicando las reglas antes mencionadas, por ejemplo $[t_4, t_3], [t_4, t_0] \rightarrow [t_4, (t_3 || t_0)]$, $[t_4, t_0], [t_4, t_2] \rightarrow [t_4, (t_0 + t_2)]$. Si seguimos realizando fragmentos con las reglas de construcción, al final obtenemos estos bloques de red de Petri, $[t_0, t_1]$, $[t_4, t_3]$, $[t_4, (t_3 || t_0)]$, $[t_4, (t_3 || t_2)]$, $[t_4, (t_0 + t_2)]$, $[t_1, (t_5 + t_6)]$, $[t_2, (t_5 + t_6)]$, $[t_3, (t_5 + t_6)]$, $[(t_3 || t_2), t_6]$, $[(t_3 || t_1), t_6]$, $[(t_1 + t_2), t_6]$, $[(t_3 || t_2), t_5]$, $[(t_3 || t_1), t_5]$, $[(t_1 + t_2), t_5]$ y $[(t_5 + t_6), t_4]$. Después de unir estos fragmentos, el resultado es el modelo de red de Petri mostrado en la figura 3.11.

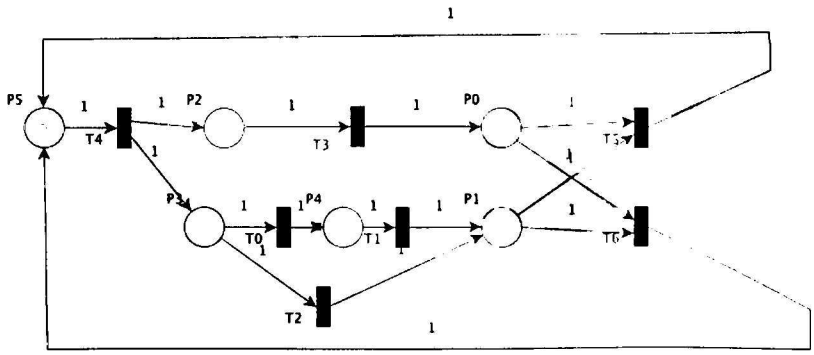


Figura 3.11. red de Petri resultado del ejemplo 3.10

Para ejemplificar la restricción de la proposición 3.8 acerca de que las transiciones (t_b , t_c) no pueden aparecer en el conjunto *ConcR*, examinemos la red de Petri que se muestra en la figura 3.12, la cual tiene dos t-invariantes de los cuales sus soportes correspondientes son, $Y_1 = \{t0, t1, t2\}$, $Y_2 = \{t3, t4\}$ y existe un lugar en esta red (P4) el cual soporta las relaciones causales *[t0, t3], [t0, t2], [t4, t2], [t4, t3]*, dos de éstas *[t0, t3] y [t4, t2]*, con una de sus transiciones dentro de un t-invariante y la otra en diferente t-invariante y las dos restantes *[t0, t2] y [t4, t3]* con ambas transiciones en el mismo t-invariante, además de las otras relaciones. Pero si aplicamos las reglas de construcción, por ejemplo a las relaciones *[t0, t3], [t0, t1]*, como *t1, t3* nunca están juntos en ningún t-invariante entonces se obtendría *[t0, (t3 + t1)]* lo cual es imposible como se puede observar en red de Petri.

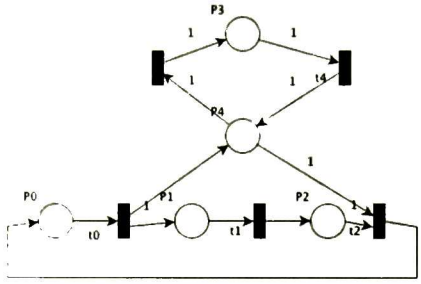


Figura 3.12. red de Petri

3.4 Ajuste al modelo

Una vez realizado el primer modelo (RP_1), podemos verificar entonces que este modelo puede reproducir la secuencia *S* de la cual fue deducido con ayuda de los t-invariantes, ya que si los t-invariantes correspondientes a RP_1 coinciden con los de nuestro análisis. Entonces esta red de Petri tiene el mismo lenguaje mostrado en *S*.

Existe el caso en que los t-invariantes de RP_1 no coinciden con los soportes de t-invariantes de nuestro análisis, este comportamiento se debe a que la secuencia *S* no proporciona información para determinar *relaciones implícitas* entre las transiciones. Esto no se debe a que *S* sea una secuencia incompleta para el sistema si no porque es imposible observar consecutivamente estas transiciones.

Definición 3.13. En una RP a salvo, la dependencia $[t_a, t_b]$ se dice implícita, si a pesar de existir un lugar entre las transiciones, el disparo de t_a no produce un marcado que habilita inmediatamente a t_b, es decir, se requiere del disparo de al menos otra transición en RP antes de disparar t_b.

Ejemplo 3.11. Para la red de Petri de la figura 3.13, existen dependencias implícitas entre las transiciones t_2, t_5 y entre las transiciones t_3, t_6. En $[t_2, t_5]$, cuando se dispara t_2, se requiere del disparo de t_4 para habilitar t_5. En situación similar para $[t_3, t_6]$, se requiere el disparo de t_4 para producir un marcado que habilite a t_6 después de disparar t_3

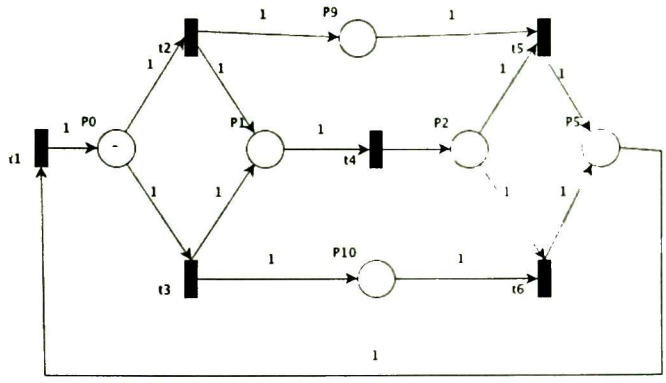


Figura 3.13. red de Petri con dependencias implícitas

Este tipo de redes con dependencias implícitas exhiben un comportamiento peculiar que dificulta generar un modelo adecuado para la secuencia *S*. podemos observar en la red de Petri de la figura 3.13 la presencia de un lugar entre las transiciones t_2 y t_5, como vimos antes al explicar la dependencia implícita.

Ahora bien, ejecutando una secuencia de transiciones *S* para esta red tenemos $S = t_2\ t_4\ t_5\ t_1\ t_2\ t_4\ t_5\ t_1\ t_3\ t_4\ t_6\ t_1\ t_2\ t_4\ t_5\ t_1\ t_3\ t_4\ t_6\ t_1\ t_3\ t_4\ t_6\ t_1$, de la cual obtenemos los siguientes conjuntos de relaciones entre las tareas $Seq = \{(t_2, t_4), (t_4, t_5), (t_4, t_6), (t_5, t_1), (t_1, t_2), (t_1, t_3), (t_3, t_4), (t_6, t_1)\}$. Para esta secuencia no se encontraron relaciones concurrentes ni comportamientos de ciclo doble; existe un lugar entre t_2 y t_5 pero no es posible observar en *S* la relación $t_2 < t_5$ y por lo tanto esta relación no se calcula con el análisis de causalidad y es imposible determinar este lugar. Sin embargo de esta secuencia es posible detectar los verdaderos t-invariantes del sistema $Y_1 = [t_2\ t_4\ t_5\ t_1]$, $Y_2 = [t_3\ t_4\ t_6\ t_1]$; esta información será muy útil para construir un modelo completo haciendo los ajustes correspondientes al anteriormente obtenido; esto se aborda enseguida.

3.4.1 Búsqueda de dependencias implícitas

Una vez que tenemos una primera aproximación del modelo (RP_1), podemos entonces verificar si los t-invariantes de dicho modelo coinciden con los obtenidos de nuestro análisis a partir de la secuencia de transiciones *S*. Si ambos coinciden entonces podemos dar como resultado RP_1, modelo que en efecto puede reproducir la secuencia presentada en *S*, pero si no es el caso existen dos posibilidades.

Caso 1. Todos los t-invariantes resultado del análisis de la secuencia *S*, coinciden con algún t-invariante en el modelo RP_1, pero éste tiene más t-invariantes que no están incluidos en el primer grupo.

Caso 2. Algunos t-invariantes resultado del análisis de la secuencia *S*, no coinciden con los encontrados en el modelo RP_1.

Analizaremos de manera separada cada uno de los casos, pero antes presentamos algunas proposiciones que surgen de la definición de dependencia implícita.

Proposición 3.11. Sean dos transiciones t_a, t_b en una RP *N*. Si existe una relación de dependencia implícita entre ellas, entonces t_a, t_b deben de aparecer juntas en al menos uno de los soportes de t-invariantes de *N*.

Demostración. Supongamos que t_a, t_b no comparten ningún t-invariante en común en el sistema, entonces tampoco existe un circuito *SE* el cual contenga ambas transiciones, lo que impide la observación de t_b en alguna de las transiciones siguientes después de ejecutar t_a, entonces no es posible encontrar un marcado intermedio que la habilite, el cual sea alcanzable por el disparo de al menos una transición, es decir no existe la relación de dependencia implícita (definición 3.13).

Ahora bien, necesitamos establecer las condiciones para determinar la relación explícita entre transiciones a partir de los resultados del análisis de la secuencia *S*. Para analizar el primer caso en que, todos los t-invariantes determinados de la secuencia *S*, coinciden con algún t-invariante en el modelo RP_1 pero este presenta más t-invariantes que no están incluidos en el primer grupo, se debe agregar un lugar entre todo aquel par de transiciones que cumpla con la siguiente regla.

$$[t_i, t_j] \leftrightarrow \neg(t_i < t_j) \wedge ((t_i \wedge t_j) \in Dist) \wedge ((t_i \wedge t_j) \in Y_x)$$

A manera de entender esta propiedad regresemos a al ejemplo 3.11 en el cual se presento la secuencia $S = t_2\ t_4\ t_5\ t_1\ t_2\ t_4\ t_5\ t_1\ t_3\ t_4\ t_6\ t_1\ t_2\ t_4\ t_5\ t_1\ t_3\ t_4\ t_6\ t_1\ t_3\ t_4\ t_6\ t_1$. A partir de esta secuencia obtenemos la información mostrada en la tabla 3.1.

Transición	*Seq*	*CausalR*	*ConcR*	*Seq´*	$\neg(t_i < t_j)$
t1	t2 t3	t2 t3			t4 t5 t6
t2	t4	t4			t1 t6 t5 t3
t3	t4	t4			t3 t6 t5 t1 t2
t4	t5 t6	t5 t6			t1 t2 t3
t5	t1	t1			t2 t4 t6 t3
t6	t1	t1			t2 t4 t5 t3

Tabla 3.1 Relaciones entre transiciones para el ejemplo 3.11

En cada columna de la tabla 3.1 se muestran las relaciones correspondientes a la transición que aparece en la primera columna hacia las demás, los conjuntos de dependencias repetitivas maximales *RdMs* encontrados a partir de *S* son $RdM_1 = \{ t_2\ t_4\ t_5\ t_1 \}$, $RdM_2 = \{ t_3\ t_4\ t_6\ t_1 \}$, y la relación $R = CausalR$ por que *Seq* no tiene elementos, de aquí la siguiente imagen muestra los grafos *G* correspondientes.

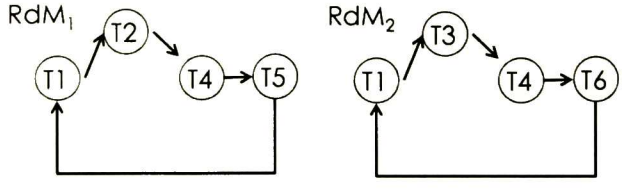


Figura 3.14. Grafos de *RdMs* para el ejemplo 3.11

Puesto que todos los grafos formados son fuertemente conexos (Sc) entonces podemos corroborar la información dicha anteriormente, los soportes de t-invariantes para esta secuencia son $<Y_1> = \{t_2\ t_4\ t_5\ t_1\}$, $<Y_2> = \{t_3\ t_4\ t_6\ t_1\}$, con la información recabada hasta el

momento procedemos a construir una primera aproximación del modelo, aplicando las reglas de construcción vistas anteriormente. Después de aplicar las reglas a las relaciones observadas en R, se obtuvieron las siguientes configuraciones, $[t_4, (t_5 + t_6)]$, $[t_1, (t_2 + t_3)]$, $[(t_2 + t_3), t_4]$, $[(t_5 + t_6), t_1]$, se estas se produce RP_1 como se muestra en la siguiente figura.

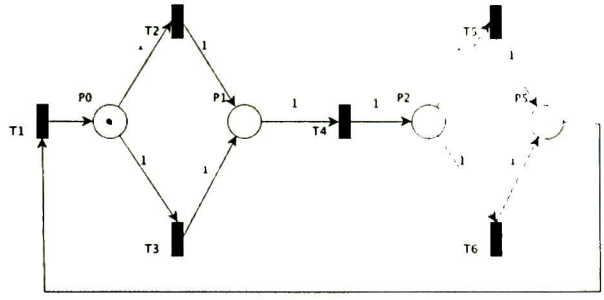


Figura 3.15. primera aproximación (RP_1) al modelo de la secuencia *S*

Caso 1. Es momento de verificar si efectivamente los t-invariantes de nuestro análisis coinciden con los t-invariantes reales de RP_1. Los t-invariantes de RP_1 son los siguientes $<Y_1'> = \{t_5, t_4, t_1, t_2\}$, $<Y_2'> = \{t_1, t_3, t_4, t_6\}$, $<Y_3'> = \{t_4, t_6, t_1, t_2\}$, $<Y_4'> = \{t_4, t_2, t_3, t_6\}$, podemos observar que nuestro análisis descubrió dos de estos t-invariantes $<Y_1'> = \{t_5, t_4, t_1, t_2\}$, $<Y_2'> = \{t_1, t_3, t_4, t_6\}$, sin embargo RP_1 tiene un comportamiento excesivo y es necesario agregar lugares al modelo para impedir este comportamiento excesivo, colocando para ello lugares que concuerden con la regla del caso 1. Los siguientes pares de transiciones cumplen con todas las restricciones en esta regla $[t_2, t_5]$, $[t_5, t_2]$, $[t_3, t_6]$, $[t_6, t_3]$, poniendo estos cuatro lugares obtenemos el modelo RP_2 de la siguiente figura

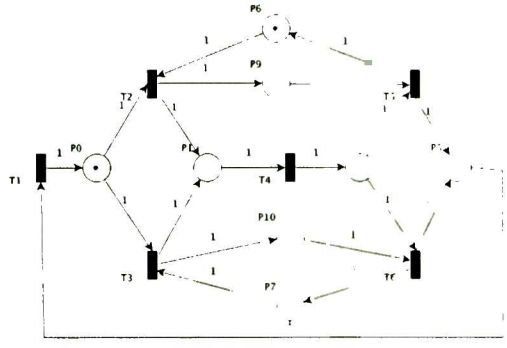


Figura 3.16. Modelo RP_2 después del ajuste

Caso 2. Cuando algunos t-invariantes Y_i resultado del análisis de la secuencia *S*, no coinciden con los encontrados en el modelo RP_1 ($C.Y_i \neq 0$). Realizaremos el siguiente procedimiento para encontrar las relaciones implícitas del sistema.

1. Calculamos la matriz de incidencia *C* del modelo RP_1.
2. Si para algunos de los t-invariantes Y_i, se cumple $C.Y_i \neq 0$, analizamos los lugares P_x correspondientes a las filas para las cuales el vector $C.Y_i$ tiene un valor diferente de cero.

3. Localizamos las transiciones de entrada y de salida para el lugar P_x.
4. Agregamos la relación (t_a, t_b) no existente en *Seq* tal que:
   t_a, t_b pertenecen al mismo $<Y_i>$
   $t_a \in Dist$ y $t_b \in P_x \cdot$ o bien $t_b \in Dist$ y $t_a \in \cdot P_x$

En el siguiente algoritmo resume el procedimiento anteriormente descrito.

Algoritmo 3.2 Encontrar las dependencias implícitas

Entrada: *C:* Matriz de incidencia de RP, Y: t-invarinates.

Salida: RP_2 con las dependencias implícitas encontradas

1. Para cada $Y_i \in Y$
   1.1 $V \leftarrow C \cdot Y_i$
   1.2 Para todo V[i] $\in V$
      Si V[i] $\neq 0$
         Entonces $P_x \leftarrow P_i$
         Para todo $(t_i, t_j) \mid \neg(t_i < t_j)$
            Si $t_i, t_j \in Y_i \wedge [(t_i \in Dist \wedge t_j \in P_x \cdot) \oplus (t_i \in \cdot P_x \wedge t_j \in Dist)]$
               Entonces agregar la relación (t_i, t_j) a RP_1

Ejemplo 3.12. Para la siguiente cadena *S* = t6 t0 t2 t5 t4 t6 t1 t2 t4 t6 t1 t2 t4 t6 t0 t2 t3 t6 t1 t2 t4 t6 t0 t5 t2 t4 t6 t0 t2 t3 t6 t0 t5 t2 t4 t6 t0 t2 t3 t6 t0 t5 t2 t4 t6 t0 t5 t2 t4 t6 t0 t2 t3 t6 t0 t5 t2 t4 t6 t0 t5 t2 t4 t6 t1 t2 t4 t6 t0 t2 t3 t6 t1 t2 t4 t6 t1 t2 t4 t6 t1 t2 t4 t6 t0 t5 t2 t4 t6 t0 t2 t5 t4 t6 t1 t2 t4 t6 t1 t2 t4 t6 t1 t2 t4 t6 t0 t5 t2 t4 t6 t0 t5 t2 t4 t6 t0 t5 t2 t4 t6 t0 t5 t2 t4 t6 t0 t5 t2 t4 t6 t0 t2 t3 t6 t0 t5 t2 t4 t6 t0 t5 t2 t4 t6 t0 t5 t2 t4 t6 t0 t2 t5 t4 t6 t1 t2 t4 t6 t0 t5 t2 t4 t6 t1 t2 t4 t6 t0 t5 t2 t4 t6 t1 t2 t4 t6 t1 t2 t4 t6 t0 t5 t2 t4 t6 t0 t2 t5 t4 t6 t0 t2 t3 t6 t1 t2 t4 t6 t0 t2 t5 t4 t6 t0 t2 t3 t6 t0 t2 t3 t6 t0 t2 t3 t6 t0 t2 t5 t4 t6 t1 t2 t4 t6 t1 t2 t4 t6 t0 t2 t5 t4 t6 t1 t2 t4 t6 t1 t2 t4 t6 t0 t5 t2 t4 t6 t0 t5 t2 t4 t6 t0 t5 t2 t4 t6 t0 t2. Se obtuvieron las relaciones entre transiciones mostradas en la tabla 3.2.

Transición	*Seq*	*CausalR*	*ConcR*	*Seq´*	$\neg(t_i < t_j)$
t0	t2 t5	t2 t5			t4 t1 t6 t3
t1	t2	t2			t0 t4 t5 t6 t3
t2	t4 t5 t3	t4 t3	t5		t1 t6 t0
t3	t3 t6	t6			t0 t4 t5 t1 t2
t4	t6	t6			t0 t1 t5 t2 t3
t5	t2 t4	t4	t2		t0 t1 t6 t3
t6	t1 t0	t1 t0			t2 t4 t5 t3

Tabla 3.2 Relaciones entre transiciones para el ejemplo 3.12

Después del análisis de las dependencias repetitivas entre las transiciones se llegó a los siguientes conjuntos de *RdMs*, RdM_1 ={t5, t4, t6, t0,t2}, RdM_2 ={t1, t2, t4, t6}, RdM_3 ={t3,

t2, t6, t0}, de donde calculamos el conjunto *Dist = {t1, t5, t3}*. Como ningún par de transiciones en este conjunto se encuentra en una relación concurrente entonces los *RdMs* son correctos. Ahora se construyen los grafos *G* correspondientes con las relaciones en *R = CausalR* ya que como se observa en la tabla 3.1 no existen elementos en *Seq*; la figura 3.14 muestra cada uno de los grafos para los conjuntos de *RdMs*.

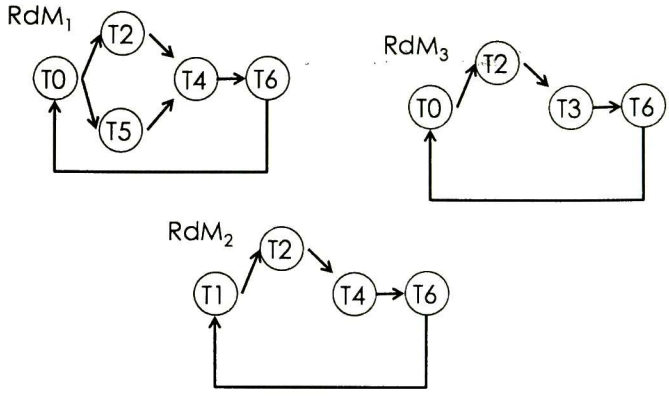


Figura 3.14. Grafos de *RdMs* para el ejemplo 3.12

Afortunadamente son grafos fuertemente conexos (Sc), por lo que tenemos los soportes de t-invariantes de la secuencia *S*, $<Y_1>$ *={t5, t4, t6, t0,t2}*, $<Y_2>$ *={t1, t2, t4, t6}*, $<Y_3>$ *={t3, t2, t6, t0}*. Con la información recabada hasta el momento procedemos a construir una primera aproximación del modelo, aplicando las reglas de construcción vistas anteriormente. Después de aplicar las reglas a las relaciones observadas en R, se obtuvieron las siguientes configuraciones, *[t2 , (t4 + t3)], [t6 , (t1 + t0)], [(t1 + t0 t2)], [(t2 || t5) t4], [(t4 + t3) . t6], [t0 , (t2 || t5)];* con estas estas se produce RP_1 como se muestra en la figura 3.15.

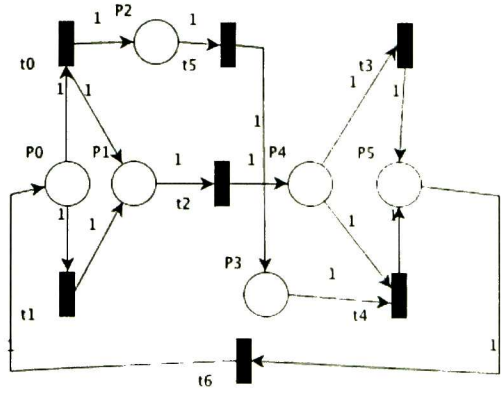


Figura 3.15. primera aproximación (RP_1) al modelo de la secuencia *S*

Es momento de verificar si los t-invariantes calculados en el análisis coinciden con los t-invariantes reales de RP_1. Los soportes de los t-invariantes de RP_1 son los siguientes: Y_1 *={t5, t4, t6, t0, t2}*, Y_2 *= {t1, t2, t3, t6};* solo el t-invariantes Y_1 *={t5, t4, t6, t0, t2}* coincide

con Y_1 . por lo tanto este comportamiento corresponde al caso 1 de la detección de dependencias implícitas. Comencemos entonces por calcular la matriz de incidencia C del modelo RP_1.

$$C = \begin{bmatrix} -1 & -1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & -1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 1 & -1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & -1 \end{bmatrix}$$

Para la matriz anterior las columnas corresponden a *t0, t1, t2, t3, t4, t5, t6* y las filas a los lugares *P0, P1, P2, P3, P4, P5,* respectivamente. Enseguida encontramos que $C.Y_2 \neq 0$ y $C.Y_3 \neq 0$; trataremos el primero de ellos.

$$C = \begin{bmatrix} -1 & -1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & -1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 1 & -1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & -1 \end{bmatrix} . Y_2 = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ -1 \\ 0 \\ 0 \end{bmatrix}$$

El lugar correspondiente a la fila en la cual el producto es diferente de cero es P3, entonces $\cdot P3 = \{t5\}, P3 \cdot = \{t4\}$.

A continuación buscamos las transiciones *(t_i, t_j)* entre las cuales debemos poner la relación faltante conforme a la regla.

Lo primero que observamos es que $t5 \notin Y_2$. por lo tanto *t5* no está en la relación *(t_i, t_j)*, pero $t4 \in Y_2$ y además $t4 \in P3 \cdot$, por lo tanto $t_j = t4$ en la relación que estamos buscando. Para encontrar a t_i buscamos relaciones de la forma $\neg(t_i < t4)$ y además para un $t_i \in Dist$ y $t_i \in Y_2$, la única t_i que cumple con todos estos requisitos es t_i = t1, entonces agregamos la arista necesaria al lugar P3 para formar la dependencia implícita *(t1, t4)*.

De igual forma tratamos con Y_3.

$$C = \begin{bmatrix} -1 & -1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & -1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 1 & -1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & -1 \end{bmatrix} . Y_3 = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

En este caso el lugar a analizar es P2, después de un procedimiento similar al descrito anteriormente se encuentra la relación implícita *(t0, t3)*. la cual se agrega a P2, para finalmente formar el modelo RP_2 mostrado en la figura 3.16 donde los arcos se indican con línea punteada.

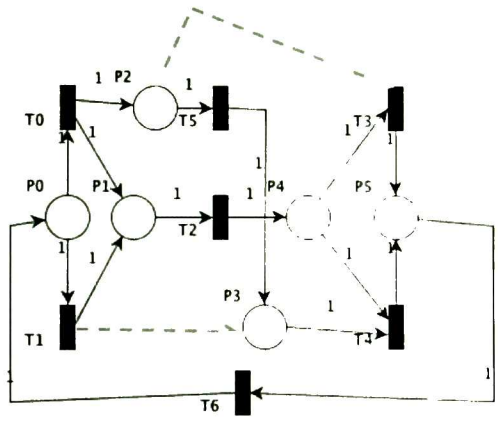


Figura 3.16. Modelo RP_2 después del ajuste

Con los arcos calculados la matriz de incidencia queda como se muestra a continuación la cual verifica $C.Y_i = 0$ para todos los t-invariantes calculados en el análisis.

$$C = \begin{bmatrix} -1 & -1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & -1 & 0 & 0 & -1 & 0 \\ 1 & 0 & 0 & 0 & 0 & -1 & 0 \\ 1 & 0 & 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 1 & -1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & -1 \end{bmatrix}$$

Capítulo 4

Implementación y Pruebas

Resumen. En este capítulo se presenta una herramienta de software que puede usarse para minado de procesos de flujo de trabajo y también para identificación de sistemas de eventos discretos. Sus principales características y el uso de la misma son mostrados, al final se muestran algunas pruebas realizadas con la herramienta, tanto en el área de minado de procesos como en un caso de estudio de un sistema de manufactura en lazo cerrado.

4.1 Descripción de la herramienta de software

A continuación se describe la implementación del método para minado de procesos presentado en el capítulo 3, la entrada al algoritmo es una secuencia de disparos de transiciones la cual proviene de un previo análisis de identificación de DES, visto en [Estrada, 2013], o bien un registro de tareas de algún proceso, para ambos casos el algoritmo arroja como resultado una red de Petri en formato XML, la cual puede ser vista en PIPE (Platform Independent Petri net Editor) y que puede reproducir la secuencia de transiciones como entrada. La implementación fue desarrollada en el IDE Netbeans 7.0.1, java jdk 1.6.0.

La interfaz de usuario se puede observar en la figura 4.1, para realizar la identificación hay que seleccionar el archivo de texto de entrada, el cual debe contener la secuencia de transiciones a identificar, después hay que seleccionar la ruta donde se alojara el archivo xml resultante de la identificación, por ultimo presionar el botón de *"start identification"*. El cuadro de texto final mostrara información acerca de la secuencia identificada y de la red de Petri que se forma después de la identificación.

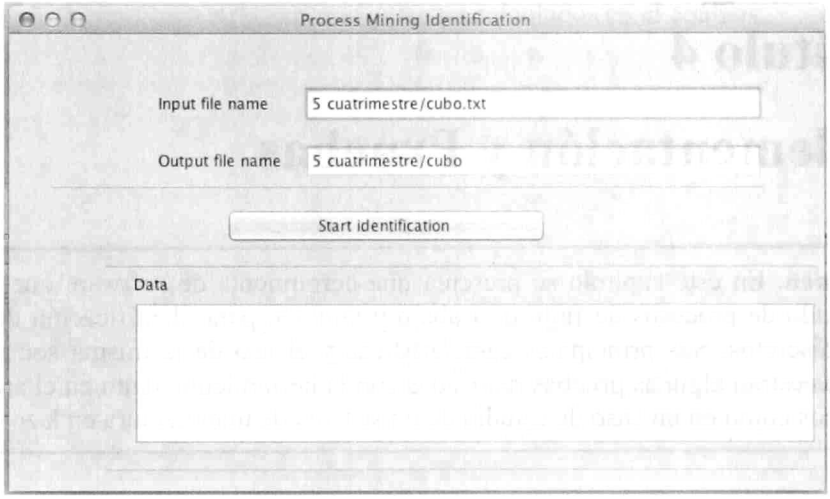
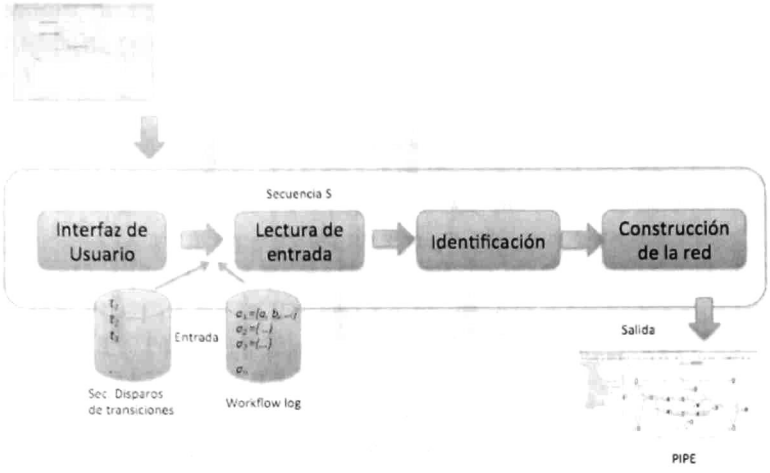


Figura 4.1. Interfaz de usuario

La figura 4.2 muestra de manera general como trabaja el software una vez que se ha presionado el botón de *"start identification"* el software recibe como entrada un archivo de texto (.txt) en el cual cada renglón representa una actividad o transición. El archivo es leído y reestructurado para obtener la secuencia de transiciones S, después se obtienen la relación entre las transiciones observadas en la secuencia y sus dependencias repetitivas para obtener los t-invariantes del sistema, por ultimo con esta información se construye el modelo de red de Petri y se presenta en formato .xml.



4.2 Arquitectura del Software.

4.2 Ejemplos de prueba para workflow logs

A continuación se incluyen varios ejemplos de minería de procesos utilizados para probar la correcta obtención de estructuras diversas en modelos de flujo de trabajo. Las secuencias fueron generadas con PIPE a partir de RPs las cuales fueron construidas mediante el método implementado. Las secuencias utilizadas como entrada al software no se muestran completas por razones de espacio.

Ejemplo 4.1 Considere la siguiente secuencia de 500 actividades generada por algún proceso de flujo se trabajo, S = T1 T3 T4 T7 T1 T4 T3 T7 T2 T6 T5 T7 T1 T4 T3 T7 T1 T4 T3 T7 T1 T4 T3 T7 T2 T6 T5 T7 T1 T4 T3 T7 T2 T5 T6 T7 T1 T3 T4 T7 T2 T5 T6 T7 T2 T5 T6 T7 T2 T5 T6 T7 T1 T3 T4 T7 T2 T6 T5 T7 T2 T5 T6 T7 T1 T3 T4 T7 T2 T6 T5 T7 T2 T5 T6 T7 T1 T3 T4 T7 T2 T6 T5 T7 T1 T3 T4 T7 T2 T5 T6 T7 T1 T4 T3 T7 T1 T3 T4 T7 T1 T4 T3 T7 T1 T3 T4 T7 T2 T5 T6 T7 T2 T5 T6 T7 T1 T3 T4 T7 T2 T5 T6 T7 T1 T3 T4 T7 T2 T5 T6 T7 T1 T4 T3 T7 T2 T6 T5 T7 T1 T4 T3 T7 T2 T5 T6 T7 T1 T3 T4 T7 T2 T5 T6 T7 T2 T6 T5 T7 T1 T4 T3 T7 T2 T6 T5 T7 T1 T4 T3 T7 T1 T3 T4 T7 T1 T3 T4 T7 T2 T5 T6 T7 T2 T5 T6 T7 T1 T4 T3 T7 T1 T4 T3 T7... Para esta secuencia el algoritmo detecta los siguientes t-invariantes: Y_1 = {T1, T3, T4, T7}, Y_2 = {T2, T5, T6, T7} y da como resultado la red que se muestra en la figura 4.3.

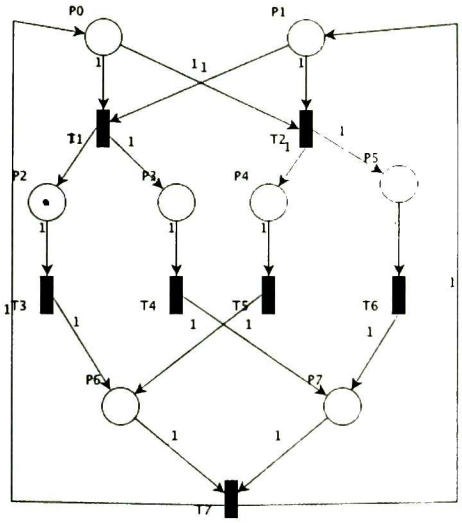


Figura 4.3 Modelo identificado para el ejemplo 4.1.

Ejemplo 4.2 Considere la siguiente secuencia de 250 actividades generada por algún proceso de flujo se trabajo, S = T0 T2 T3 T1 T2 T5 T4 T1 T5 T2 T4 T0 T2 T3 T1 T5 T2 T4 T0 T2 T3 T1 T5 T2 T4 T0 T2 T3 T0 T2 T3 T1 T5 T2 T4 T1 T5 T2 T4 T0 T2 T3 T0 T2 T3 T1 T5 T2 T4 T0 T2 T3 T1 T5 T2 T4 T1 T2 T5 T4 T0 T2 T3 T1 T5 T2 T4 T0 T2 T3 T1 T5 T2 T4 T0 T2 T3 T1 T5 T2 T4 T1 T5 T2 T4 T1 T2 T5 T4 T0 T2 T3 T T2 T3 T1 T2 T5 T4 T0 T2 T3 T1 T5 T2 T4 T1 T5 T2 T4 T1 T5 T2 T4 T0 T2 T3 T1 T2 T5 T4 T1 T5 T2 T4 T1 T2 T5 T4 T0 T2 T3 T0 T2 T3 T1 T5 T2 T4 T0 T2 T3 T1 T5 T2 T4 T0 T2 T3 T0 T2 T3 T0 T2 T3 T0 T2 T3 T1 T2 T5 T4 T0 T2 T3 T0 T2 T3 T1 T5 T2 T4 T1 T2 T5 T4 T1 T5 T2 T4 T0 T2 T3 T1 T5 T2 T4 T1 T2 T5 T4 T0 T2 T3 T1 T2 T5 T4 T0 T2 T3 T1 T5 T2 T4 T0 T2 T3 T1 T2 T5 T4 T1 T5 T2 T4 T0 T2 T3 T1 T2 T5 T4 T1 T5 T2 T4 T1 T5 T2 T4 T0 T2 T3 T0 T2 T3 T1 T2 T5 T4 T1 T5 T2 T4 T0 T2 T3 T1 T2 T5 T4 T0 T2. Para esta secuencia el algoritmo detecta los siguientes t-invariantes: Y_1= {T0, T2, T3}, Y_2= {T1, T2, T5, T4} y da como resultado la red que se muestra en la figura 4.4.

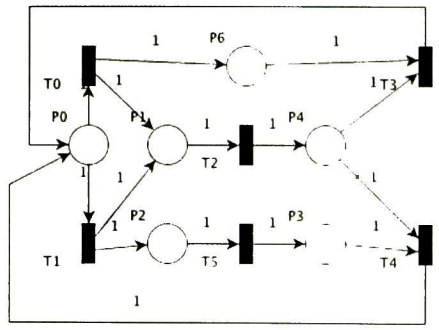


Figura 4.4. Modelo identificado para el ejemplo 4.2.

Nótese que el algoritmo es capaz de detectar la presencia del lugar P6; el cual tiene una relación de dependencia implícita entre las tareas T0, T3.

Ejemplo 4.3 Considere la siguiente secuencia de 1500 actividades generada por algún proceso de flujo se trabajo, por cuestiones de espacio solo se mostrara parte de la secuencia, S = T16 T1 T2 T4 T3 T5 T8 T6 T10 T11 T12 T15 T16 T14 T2 T3 T5 T9 T3 T4 T6 T5 T8 T10 T17 T2 T4 T7 T3 T5 T8 T17 T2 T3 T5 T8 T4 T6 T10 T17 T2 T4 T7 T3 T5 T8 T11 T13 T15 T16 T1 T2 T4 T3 T7 T5 T9 T3 T5 T9 T3 T5 T8 T11 T12 T15 T16 T14 T2 T4 T6 T3 T10 T5 T9 T3 T5 T9 T3 T5 T8 T11 T13 T15 T16 T1 T2 T3 T4 T6 T5 T9 T3 T10 T5 T8 T11 T12 T15 T16 T1 T2 T4 T7 T3 T5 T8 T17 T2 T4 T3 T6 T5 T8 T10 T17 T2 T4 T3 T6 T5 T8 T10 T11 T12 T15 T16 T1 T2 T4 T7 T3 T5 T9 T3 T5 T9 T3 T5 T8 T17 T2 T3 T5 T9 T3 T5 T4 T8 T7 T11 T12 T15 T16 T1 T2 T4 T7 T3 T5 T8 T11 T12 T15 T16 T1 T2 T4 T7 T3 T5 T8 T11 T12 T15 T16 T14 T2 T4 T6 T10 T3 T5 T8 T11 T13 T15 T16 T14 T2 T4 T3 T7 T5 T9 T3 T5 T8 T17 T2 T4 T7 T3 T5 T8 T17 T2 T3 T5 T9 T4 T6 T10 T3 T5 T8 T17 T2 T3 T4 T6 T5 T8 T10 T17 T2 T3 T5 T9 T3 T4 T6 T5 T10... . Para esta secuencia el algoritmo detecta los siguientes t-invariantes: Y_1 = {T14, T16, T2, T4, T3, T5, T8, T11, T15, T13, T6, T10} , Y_2 = {T14, T16, T2, T4, T3, T5, T8, T11, T15, T13, T7}, Y_3 = {T1, T16, T2, T4, T3, T5, T8, T11, T12, T15, T6, T10} Y_4 = {T1, T16, T2, T4, T3, T5, T8, T11, T12, T15, T7}, Y_5 = {T6, T2, T4, T10, T3, T5, T8, T17}, Y_6 = {T17, T2, T4, T3, T5, T8, T7}, Y- = {T9, T3, T5} y da como resultado la red que se muestra en la figura 4.5.

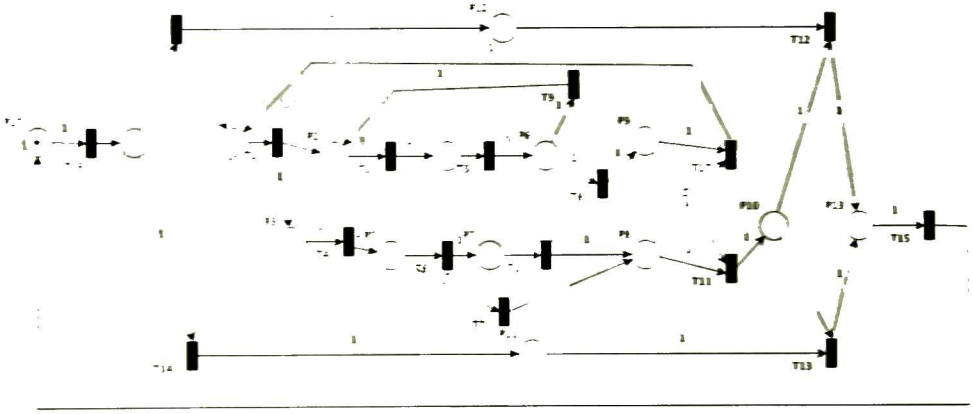


Figura 4.5. Modelo identificado para el ejemplo 4.3.

A pesar de que la secuencia del ejemplo anterior contienen invariantes que están dentro de otro invariante de la red, tal es el caso de Y_7 = {T9, T3, T5} el cual el parte de otros t-invariantes en el sistema, el método es capaz de detectar cada uno de los t-invariantes que de la red, a demás de detectar las dependencias implícitas entre T1, T12 y T14, T13, lo que restringe el comportamiento de la red.

Ejemplo 4.4 Considere la siguiente secuencia de 1000 actividades generada por algún flujo de trabajo. S = T1 T2 T4 T5 T3 T1 T4 T5 T6 T4 T7 T5 T2 T3 T1 T4 T2 T3 T5 T8 T6 T4 T5 T1 T2 T7 T4 T5 T3 T1 T8 T4 T5 T2 T3 T6 T7 T4 T5 T1 T4 T8 T2 T5 T6 T3 T4 T7 T5 T8 T6 T4 T5 T1 T7 T2 T3 T8 T4 T5 T1 T4 T5 T6 T7 T2 T8 T4 T3 T5 T6 T4 T7 T8 T5 T1 T2 T4 T3 T5 T6 T7 T8 T4 T5 T1 T4 T5 T6 T4 T2 T3 T7 T8 T5 T1 T2 T4 T3 T5 T1 T4 T2 T3 T5 T6 T4 T5 T7 T1 T4 T5 T2 T8 T3 T1 T4 T2 T5 T3 T6 T4 T7 T8 T5 T1 T4 T5 T6 T4 T5 T7 T8 T2 T3 T1 T4 T5 T6 T7 T4 T8 T5 T2 T6 T3 T7 T8 T4 T5 T1 T4 T2 T3 T5 T6 T4 T7 T5 T8 T1 T2 T3 T4 T5 T1 T2 T4 T5 T3 T1 T4 T2 T5 T3 T1.... para esta secuencia

el algoritmo detecta los siguientes t-invariantes: Y_1= {T1 T2 T4 T5 T3}, Y_2= {T6 T4 T5 T7 T8} y arroja como resultado el modelo de la figura 4.6.

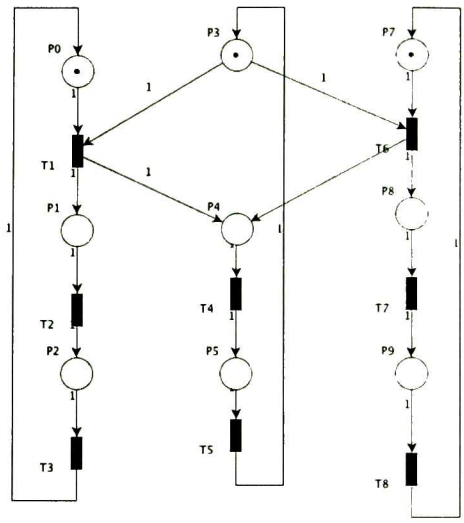


Figura 4.6. Modelo identificado para el ejemplo 4.4

La imagen anterior muestra un comportamiento de recursos compartidos en el que solo una de las dos componentes repetitivas puede ejecutarse a la vez.

Ejemplo 4.5 Considere la siguiente secuencia de 500 actividades generada por algún flujo de trabajo, S = T1 T4 T3 T4 T1 T4 T3 T4 T3 T5 T2 T3 T6 T5 T4 T8 T1 T5 T6 T5 T8 T5 T6 T7 T8 T7 T2 T1 T4 T8 T3 T2 T5 T6 T1 T4 T5 T1 T6 T2 T1 T7 T2 T3 T6 T2 T5 T1 T2 T8 T5 T3 T8 T7 T8 T4 T5 T6 T5 T8 T5 T1 T2 T3 T6 T5 T4 T1 T2 T6 T7 T8 T1 T4 T5 T1 T6 T5 T4 T8 T5 T3 T6 T7 T8 T4 T1 T7 T8 T4 T7 T1 T6 T7 T2 T3 T6 T2 T7 T1 T8 T4 T5 T6 T3 T4 T5 T3 T2 T3 T6 T2 T1 T7 T8 T4 T5 T1 T6 T5 T2 T3 T6 T5 T4 T8 T5 T8 T7 T8 T7 T6 T3 T7 T2 T6 T3 T4 T3 T2 T1 T2 T1 T4 T7 T8 T7 T3 T6 T4 T7 T1 T4 T1 T8 T7 T6 T5 T4 T1 T4 T3 T4 T6 T3 T5 T6 T2 T5 T8 T1 T2 T7 T8 T3 T2 T1 T5 T4 T6 T7 T3 T6 T2 T3 T7 T6 T4 T5 T6 T1 T5 T4 T3 T8 T7 T6 T5 T2 T1 T6 T2 T3 T7 T8 T2 T1 T4 T5 T3 T2 T8 T3 T7 T6 T2 T3 T7 T6 T2 T7 T3..., para esta secuencia el algoritmo encuentra los siguientes t-invariantes Y_1 = {T1 T4}, Y_2 = {T1 T2}, Y_3 = {T4 T3}, Y_4 = {T3 T2} Y_5 = {T5 T6}, Y_6 = {T5 T8}, Y_7 = {T6 T7}, Y_8 = {T8 T7} y arroja como resultado el modelo de la figura 4.7.

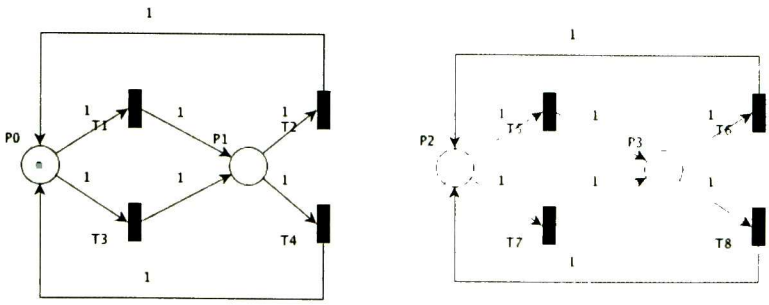


Figura 4.7 Modelo identificado para el ejemplo 4.5.

Esta secuencia tiene dos comportamientos interesantes el primero es que forma dos componentes desconectadas, a pesar de eso el método es capaz de identificarla, con una pequeña observación a la secuencia notamos el segundo comportamiento; el conjunto dependencia repetitiva para cada una de las tareas consiste solo de ellas mismas, por lo tanto sus dependencias repetitivas máximas, permanecerán sin cambio. Sin embargo el método encuentra las componentes repetitivas (t-invariantes) del sistema gracias a las uniones entre RdM, como vio con anterioridad.

Ejemplo 4.6 Considere la siguiente secuencia de 500 actividades generada por algún flujo de trabajo, S = T0 T7 T2 T1 T4 T6 T0 T1 T7 T2 T4 T5 T0 T1 T7 T2 T4 T5 T0 T7 T1 T2 T4 T5 T0 T3 T2 T1 T4 T6 T0 T2 T7 T1 T4 T6 T0 T3 T2 T1 T4 T6 T0 T3 T1 T2 T4 T6 T0 T7 T2 T1 T4 T5 T0 T7 T1 T2 T4 T5 T0 T3 T1 T2 T4 T5 T0 T3 T2 T1 T4 T5 T0 T7 T1 T2 T4 T5 T0 T7 T1 T2 T4 T6 T0 T2 T3 T1 T4 T5 T0 T1 T2 T7 T4 T5 T0 T1 T3 T2 T4 T6 T0 T1 T3 T2 T4 T5 T0 T2 T3 T1 T4 T5 T0 T7 T1 T2 T4 T5 T0 T1 T3 T2 T4 T5 T0 T3 T2 T1 T4 T6 T0 T3 T1 T2 T4 T5 T0 T1 T3 T2 T4 T5 T0 T2 T1 T7 T4 T5 T0 T3 T2 T1 T4 T5 T0..., para esta secuencia el algoritmo encuentra los siguientes t-invariantes: Y_1 = {T0, T4 ,T2 , T1 , T6 , T7}, Y_2 = {T0, T4 ,T2 , T1 T5 , T7}, Y_3 = {T0, T4 ,T2 , T1 , T6 , T3}, Y_4 = {T0, T4 ,T2 , T1 T5 , T3} y arroja como resultado el modelo de la figura 4.8.

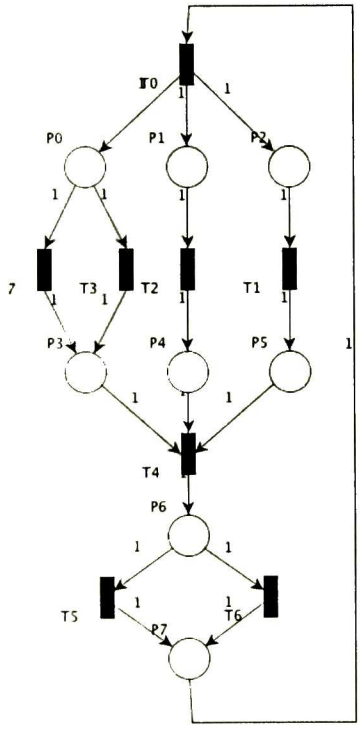


Figura 4.8 Modelo identificado para el ejemplo 4.6

Para esta secuencia al igual que en la anterior las uniones entre RdMs nos ayudan a detectar los puntos de bifurcación que se encuentran en el lugar P0 y P6, los cuales determinan los 4 t-invariantes del sistema.

4.3 Ejemplo de prueba para Identificación de DES

Para probar las pruebas y ejemplos presentados en esta parte del capítulo, se hiso uso de algunas de las herramientas en el LURPA. Se utilizo un *Mechatronics Estándar System (MMS) Figura 4.9*, de la marca Bosch; este es una maquina de ensamblaje que consta de 4 estaciones. La maquina recibe varias piezas de engranaje de diferente material, el objetivo de la maquina es colocar o remover empaques según sea el caso y ordenar los engranajes en almacenes según el tipo de material.

Figura 4.9 Mechatronics Standard System

Para obtener la información del MMS, el controlador (PLC) se comunica con la planta vía Etherent [Roth, 2010c]. y los datos son recolectados con un programa en Python que permite a la computadora obtener las secuencias de entrada a través del protocolo de comunicación Modbus.

La MMS tienes muchos modos de operación que permiten al usuario interactuar con la maquina. Para propósitos de este trabajo te tomara como referencia el caso de estudio visto en [Estrada, 2013], en el cual solo la cuarta estación es tomada en cuanta para el proceso de identificación, esta parte del proceso se encarga de recibir las piezas de engranaje y colocarlas en el almacén correcto según su tipo de materias. Para la recolección de los datos se utilizo un escenario donde las piezas de engranaje están ordenadas por material en alguna de las bandejas disponibles. Una secuencia de vectores de entrada-salida de 63,797 es tomada de la ejecución de la maquina. Los índices de las entradas y salidas correspondientes son 16 entradas 3B11, 4S24, 4S23, 4S22, 4S21, 4S20, 4S17, 4B16, 4B15, 4B14, 4B13, 4B12, 4B11, 4B10, 4B07, 4S06) y 6 salidas (4Y11, 4Y10, 4Y07, 4Y06, 4K05, 4K04).

Una vez analizada la secuencia de vectores entrada salida por el método de [Estrada, 2013], se generan las funciones de disparo para cada transición y se forman los fragmentos observables de la red de Petri mostrados en la figura 4.10.

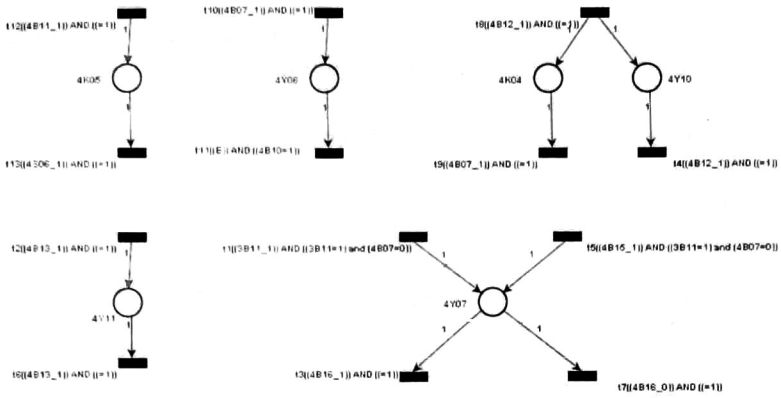


Figura 4.10 Comportamiento observado encontrado para la MMS

A partir de la secuencia de vectores entrada-salida w y de las transiciones encontradas se obtiene una secuencias de disparos de transiciones S de 6,240 transiciones, la cual no se mostrara debido a su gran tamaño.

Una vez que tenemos dicha secuencia podemos aplicar el mismo método utilizado anterior mente para la identificación de flujos de trabajo, pero ahora para encontrar la parte no observable del modelo encontrado por el método de [Estrada, 2013] anteriormente, el algoritmo encuentra para esta secuencia dos t-invariantes: Y_1= {t14 t1 t2 t3 t4 t5 t6 t8 t9 t10 t11 t12 t13}, Y_2= {t7 t1 t2 t3 t4 t5 t6 t8 t9 t10 t11 t12 t13}. El modelo no observable que se obtuvo de la secuencia *S*, se presenta en la figura 4.11.

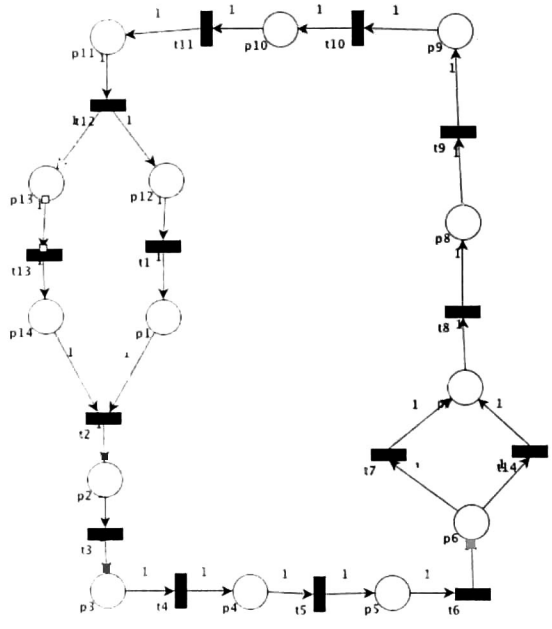
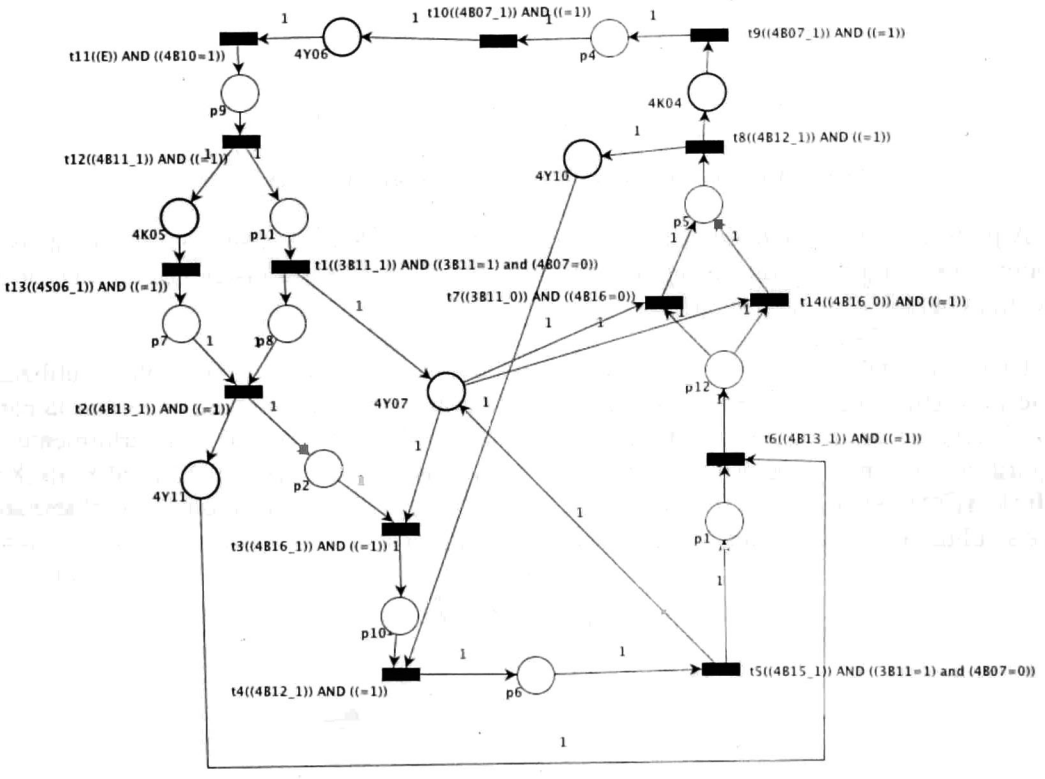


Figura 4.10 Modelo no observable

Un modelo completo del sistema es mostrado en la figura 4.11, este muestra el

comportamiento del sistema: el transportador llega con una pieza y la pinza rotatoria baja para tomarla. Una vez que la pinza esta abajo, toma la pieza y comienza a subir de nuevo. Una vez que la pinza sube, comienza a moverse a la derecha. Una vez en la posición final a la derecha, baja de nuevo para depositar la pieza en otro transportador. Una vez que la pieza es colocada en el transportador, la pinza sube nuevamente y va hacia la izquierda, para regresar a la posición inicial.



4.11 Modelo final para la estación 4 del MMS

4.4 Comentarios sobre las pruebas

La implementación de la herramienta de software para el método de minería de procesos mostrado en el capítulo 3, ha sido probada para dos casos de estudio, el primero para la identificación de procesos de flujo de trabajo y un segundo para la identificación de sistemas de eventos discretos. En algunos de los ejemplos se muestra como el algoritmo puede encontrar las verdaderas dependencias repetitivas entre las tareas, además de encontrar las dependencias implícitas que pueden existir en la secuencia S.

Conclusión

En esta tesis se aborda el problema de la minería de procesos de flujo de trabajo, donde a partir de secuencias de tareas correspondientes a ejecuciones de procesos, se obtienen modelos de redes de Petri (RP) a salvo.

Se definió e implementó un método que procesa las secuencias y construye sistemáticamente una RP que representa los procesos; en particular, el método trata con una sola secuencia *S* formada por la concatenación de múltiples secuencias de tareas. La RP obtenida; donde cada tarea es representada por una sola transición permite la ejecución de la secuencia *S*.

El método propuesto permite encontrar a partir de *S* los soportes de t-invariantes y con ayuda de un conjunto reglas de construcción se obtiene una RP_1 que reproduce la secuencia de disparos presentada en *S*. Sin embargo esta red puede aceptar además otras secuencias. Para tratar con este problema se propone una técnica de verificación y ajuste del modelo basado en los t-invariantes la cual permite la detección de dependencias implícitas entre ciertas transiciones. El modelo ajustado RP_2 acepta la secuencia S, excediendo eventualmente la representación de sub-secuencias iteradas en S. Esto es inherente a un modelo con componentes repetitivas.

La implementación de los algoritmos derivados del método permitió hacer pruebas exhaustivas con modelos de complejidad diversa, los cuales incluyeron estructuras anidadas de ciclos, concurrencia y dependencias implícitas. Durante las pruebas se utilizaron secuencias de longitudes diferentes. Adicionalmente, se realizó un caso de estudio que trata con un sistema de manufactura real, donde los procesos son de flujo de material.

Los resultados obtenidos pueden ser comparados ahora con los representativos del área de minería de procesos. Retomando la tabla presentada en capitulo uno, se incluyen las características de nuestro método; esta tabla se incluye más abajo. Como puede apreciarse se han propuesto soluciones para algunos aspectos no cubiertos en otros trabajos.

El presente trabajo contribuye modestamente al estudio de la ingeniería de reversa de procesos de flujo de trabajo. Existen algunos problemas abiertos que se inclinan a relajar hipótesis para tratar con problemas reales; estos representan alternativas retadoras para extender y mejorar los resultados obtenidos. A continuación se listan algunos de ellos.

Abordar el problema donde la secuencia de tareas no es completa. Un enfoque probabilístico se intuye adecuado para abordar este planteamiento.

Encontrar modelos reducidos equivalentes después de encontrar las dependencias implícitas entre las tareas (lugares innecesarios en el modelo final).

Mejorar los algoritmos propuestos a manera de aumentar su eficiencia.

Criterio de comparación / Técnica de Process Mining	Ruido	Minería de ciclos	Diferentes perspectivas	Visualizar resultados	Procesos concurrentes	Dependencia implícita
[Agrawal, 1998]		X		X	X	
[Cook, 2004]				X	X	
[Tapia, 2013]		X	X	X	X	X
[van der Aalst, 2004] algoritmo alfa				X		
[van der Aalst, 2005]	X	X		X	X	
[van der Aalst, 2004] dependencia/ frecuencia				X		
[Wang, 2012]		X		X	X	X
[Wang, 2006]		X		X	X	X
[APE 2013]		X	X	X	X	

Tabla Comparativa entre técnicas de minado de procesos

Referencias

[Agrawal, 1998] Rakesh Agrawal, Dimitrios Gunopulo, Frank Leymann, "Mining Process Models from Workflow Logs, 1988.

[Angluin, 1988] D. Angluin, "Queries and Concept Learning", *Machine Learning*, 2(4), pp. 319-342, 1988.

[Cabasino, 2009] M. P. Cabasino, "Diagnosis and Identification of Discrete Event Systems using Petri nets", *Ph. D. Thesis*, University of Cagliari, Mar 2009.

[Cook, 1998] Jonathan E. Cook and Alexander L. Wolf. Discovering models of software processes from event-based data. ACM Trans. Softw. Eng. Methodol., 7(3):215–249, July (1998).

[Cook, 2004] J. E. Cook, Z. Du, C. Liu, A. Wolf, "Discovering models of behavior for concurrent workflows", *Computers in Industry*, 53(3), pp. 297-319, 2004. Doi:10.1016/j.compind.2003.10.005.

[Estrada, 2009] A. P. Estrada-Vargas, "Identification of Concurrent Discrete Event Systems from Input-Output Sequences", *Master Thesis*, Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional, Unidad Guadalajara, Aug. 2009.

[Estrada, 2010a] A.P. Estrada-Vargas, E. López-Mellado, J.J. Lesage. "A Comparative Analysis of Recent Identification Approaches for Discrete-Event Systems", *Mathematical Problems in Engineering*, 2010, Hindawi. Doi:10.1155/2010/453254

[Estrada, 2010b] A.P. Estrada-Vargas, E. Lopez-Mellado, J-J. Lesage. "An Identification Method for PLC-based Automated Discrete Event Systems" *Proc. of the IEEE Int. Conf. on Decision and Control*, pp.6740-6746, Atlanta, USA, Dec. 2010.

[Estrada, 2011] A. P Estrada-Vargas, J.-J. Lesage, E. López-Mellado, "Stepwise Identification of Automated Discrete Manufacturing Systems", *16th IEEE Int. Conf. on Emerging Technologies and Factory Automation*, Toulouse, France, Sep. 2011.

[Estrada, 2012] A. P. Estrada-Vargas, J.-J. Lesage, E. López-Mellado, "Identification of Industrial Automation Systems: Building Compact and Expressive Petri Net Models from Observable Behavior", Proc. of American Control Conference, pp. 6095 - 6101, Montréal (Canada), Jun. 2012.

[Estrada, 2013] A. P. Estrada-Vargas, J.-J. Lesage, E. López-Mellado, "Identification of Partially Observable Discrete Event Manufacturing Systems", ETFA 2013, Calgliary (Italy), Jun. 2012.

[Esparza, 2011] Javier Esparza , Martin Leucker Maximilian Schlund, "Learning Workflow Petri Nets", *Petri Nets 2010 conference*, Garching, Germany. 2011.

[Guia, 2005] A. Giua and C. Seatzu, "Identification of free-labeled Petri nets via integer programming", Proc. of the 44th IEEE Conf. on Decision and Control, Seville, Spain, Dec. 2005

[Greco, 2004] Gianluigi Greco, Antonella Guzzo, Luigi Pontieri, and Domenico Sacca`. Mining expressive process models by clustering workflow traces. In Honghua Dai, Ramakrishnan Srikant, and Chengqi Zhang, editors, Advances in Knowledge Discovery and Data Mining, volume 3056 of Lecture Notes in Computer Science, chapter 8, pages 52–62. Springer Berlin / Heidelberg, Berlin, Heidelberg, 2004.

[Gold, 1967] E.M. Gold, "Language Identification in the Limit", *Information and Control*, 10, pp. 447-474, 1967.

[Klein, 2005a] S. Klein, L. Litz,J.-J. Lesage, "Fault detection of Discrete Event Systems using an identification approach", *16th IFAC World Congress*, CDROM paper n°02643, 6 pages, Praha(CZ), Jul. 2005.

[Klein, 2005b] S. Klein, "Identification of Discrete Event Systems for Fault Detection Purposes", *Ph. D. Thesis*, Ecole Normale Supérieure de Chachan, Oct. 2005.

[Meda, 1998] M.E. Meda, "DES identification using Interpreted Petri nets", *Int. Symposium on Robotics and Automation*, pp. 353-357, Saltillo, Mexico, Dec. 1998.

[Meda, 2000] M. Meda, A. Ramírez, E. López "Asymptotic Identification for DES", *IEEE Conf. on Decision and Control*, Sydney, Australia, pp. 2266-2271, Dec. 2000.

[Meda, 2000] M. Meda, A. Ramírez, E. López, "Behavioral properties for the control of discrete event systems modeled by interpreted Petri nets", *IEEE Int. Conf. on Systems, Man, and Cybernetics*, Nashville, USA, pp. 2150-2155, Oct. 2000.

[Meda, 2001] M. Meda, E. López, "A passive method for on-line identification of discrete event systems", *IEEE Int. Conf. on Decision and Control*, Orlando, Florida, USA pp. 4990-4995, Dec. 2001.

[Meda, 2002a] M. E. Meda-Campaña, "On-line Identification of Discrete Event Systems: Fundamentals and Algorithms for the Synthesis of Petri Net Models", *Ph. D. Thesis,* Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional, Unidad Guadalajara, Nov. 2002.

[Meda, 2002b] M. Meda, E. López, "Incremental Synthesis of Petri Net Models for Identification of Discrete Event Systems", *IEEE Conf. on Decision and Control*, Las Vegas, USA, pp.805-810, Dec. 2002.

[Meda, 2003] M. Meda, E. López, "Required Transition Sequences for DES identification", *IEEE Conf. on Decision and Control*, Maui, Hawaii USA pp. 3778-3782, Dec. 2003.

[Meda, 2005] M. Meda, E. López, "Identification of concurrent Discrete Event Systems Using Petri Nets", *IMACS 2005 World Congress*, Paris, France, pp. 1-7, Jul. 2005.

[Medeiros, 2005] A.K.A Medeiros, B.F van Dongen, W.M.P van der Aalst, A.J.M.M. Weijters, "Process Mining for Ubiquitous Mobile Systems: An Overview and a

Concrete Algorithm", Ubiquitous Mobile Information and Collaboration Systems, pp. 151-165, 2005.

[Roth, 2009] M. Roth, J.-J. Lesage, L. Litz, "An FDI Method for Manufacturing Systems Based on an Identified Model", Proc. of IFAC Symposium on Information Control Problems in Manufacturing (INCOM 2009), Moscow, Russia, pp. 1389-1394, June 2009

[Roth, 2010a] M. Roth, "Identification and Fault Diagnosis of Industrial Closed-Loop Discrete Event Systems", *Ph. D. Thesis*, Technische Universität Kaiserslautern, Ecole Normale Supérieur de Cachan, Oct. 2010.

[Roth, 2010b] M. Roth, J.-J. Lesage, L. Litz, "Black-box identification of discrete event systems with optimal partitioning of concurrent subsystems", *Proc. of the 2010 American Control Conf.*, Baltimore, USA, pp. 2601-2606, Jun.-Jul. 2010.

[Roth, 2010c] M. Roth, J.-J. Lesage, L. Litz, "Identification of Discrete Event Systems, implementation issues and model completeness", 7th Int. Conf. on Informatics in Control Automation and Robotics, Funchal, Portugal, pp. 73-80, Jun. 2010

[Roth, 2011] M. Roth, J.-J. Lesage, L. Litz, "The concept of residuals for fault localization in discrete event systems", *Control Engineering Practice*, 19(9), pp. 978-988.

[Roth, 2012] M. Roth, S. Schneider, J.-J. Lesage, L. Litz, "Fault detection and isolation in manufacturing systems with an identified discrete event model", *Int. Journal of Systems Science*, 43(10), pp. 1826-1841. Doi:10.1080/00207721.2011.649369.

[van der Aalst, 1998] W. M. P. van der Aalst. The application of Petri nets to workflow management. The Journal of Circuits, Systems and Computers, 8(1):21–66, 1998.

[van der Aalst, 2001] W. van der Aalst, T. Weijters, L. Maruster, "Process Mining Discovering Workflow Models from Event-Based Data", Proceedings of the ECAI Workshop on Knowledge Discovery and Spatial Data, 283-290, 2001.

[van der Aalst, 2004] W. van der Aalst, T. Weijters, L. Maruster, "Workflow Mining: Discovering Process Models from Event Logs", *IEEE Trans. on Knowledge and Data Engineering*, 16(9), Sep. 2004.

[van der Aalst, 2004a] van der Aalst, W.M.P. (2004a), "Process mining: a research agenda", Computers in Industry, Vol. 53, pp. 231-44.

[van der Aalst, 2005a] van der Aalst, W.M.P., Alves de Medeiros, A.K. and Weijters, A.J.M.M. (2005a), "Genetic process mining", in Ciardo, G. (Ed.), Applications and Theory of Petri Nets, Springer Verlag, Heidelberg.

[Wen, 2006] Lijie Wen, Jianmin Wang, and Jiaguang Sun. Detecting implicit dependencies between tasks from event logs. pages 591–603. 2006.

[Wen, 2007] Lijie Wen, Wil M. Aalst, Jianmin Wang, and Jiaguang Sun. Mining process models with non-free-choice constructs. Data Min. Knowl. Discov., 15(2):145–180, October 2007.

[Wen, 2009] Lijie Wen, Jianmin Wang, Wil van der Aalst, Biqing Huang, and Jiaguang Sun. A novel approach for process mining based on event types. Journal of Intelligent Information Systems, 32(2):163–190, April 2009.

[Wang, 2011] Dongyi Wang, JiDong Ge, Hao Hu, Bin Luo: Process mining algoritmh based on Event Type. DASC 2011. 1144-1151.

[Wang, 2012] Dongyi Wang, JiDong Ge, Hao Hu, Bin Luo, LiGuo Huang: Discovering process models from event multiset. Expert Syst. Appl. 39(15): 11970-11978 (2012).

[Xie, 2011] H. Y. Hu, J. N. Xie and H. Hu, A novel approach for mining stochastic process model from workflow logs, Journal of Computational Information Systems 9(2011), 3113-3126.

CENTRO DE INVESTIGACIÓN Y DE ESTUDIOS AVANZADOS DEL I.P.N.
UNIDAD GUADALAJARA

El Jurado designado por la Unidad Guadalajara del Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional aprobó la tesis

Síntesis de Redes de Petri para Minería de Procesos Basada en la Inferencia de T-Invariantes

del (la) C.

Tonatiuh TAPIA FLORES

el día 26 de Agosto de 2013.

Dr. Luis Ernesto López Mellado
Investigador CINVESTAV 3C
CINVESTAV Unidad Guadalajara

Dr. Félix Francisco Ramos Corchado
Investigador CINVESTAV 3B
CINVESTAV Unidad Guadalajara

Dr. Antonio Ramírez Treviño
Investigador CINVESTAV 3A
CINVESTAV Unidad Guadalajara

Dr. Mario Angel Siller González Pico
Investigador CINVESTAV 2C
CINVESTAV Unidad Guadalajara