



UNIVERSIDAD NACIONAL DE CÓRDOBA  
FACULTAD DE CIENCIAS EXACTAS FÍSICAS Y  
NATURALES

PROYECTO INTEGRADOR  
INGENIERÍA EN COMPUTACIÓN

---

**Determinación automática del control de  
una Red de Petri**

---

*Autor:*

NAVARRO, Matias Alejandro  
38730347  
[matias.navarro@mi.unc.edu.ar](mailto:matias.navarro@mi.unc.edu.ar)

*Director:*

Ph.D. Ing Micolini, Orlando

*Co-director:*

Ing. Ventre, Luis Orlando

Diciembre 2020



**Determinación automática del control de una Red de Petri***Resumen*

El contenido de este documento abarca las motivaciones, objetivos, detalles de implementación y demás datos pertinentes al desarrollo del Proyecto Integrador para la carrera de Ingeniería en Computación. El mismo se estructura en cuatro grandes partes: introducción del proyecto, marco teórico, desarrollo de la investigación, conclusiones obtenidas y trabajo a futuro.

El tema a abarcar es principalmente el análisis de redes de Petri que presentan estados de deadlock, teniendo como objetivo desarrollar y documentar un algoritmo que permita el control de las mismas, restringiendo el alcance de estos estados.



## *Agradecimientos*

Muchas gracias a mi familia, por el apoyo incondicional a lo largo de todos estos años de estudio. Este proyecto no hubiera sido posible sin el soporte, la confianza, la supervisión y el empeño de nuestros directores, Ph.D. Ing Micolini, Orlando e Ing. Ventre, Luis Orlando.

Un especial agradecimiento a mis amigos y todas las personas que tuvimos el placer de conocer durante estos años de carrera.

Agradezco al Laboratorio de Arquitectura de Computadoras, y a todo su personal, por las oportunidades y enseñanzas compartidas.

Finalmente, agradezco a la Facultad de Ciencias Exactas Físicas y Naturales de la Universidad Nacional de Córdoba por la oportunidad de realizar esta carrera de grado.



# Índice general

<b>Agradecimientos</b>	<b>V</b>
<b>1. Introducción</b>	<b>1</b>
1.1. Motivación e importancia del proyecto . . . . .	1
1.2. Estado del arte . . . . .	1
1.2.1. Estrategias de manejo de deadlock . . . . .	2
1.2.1.1. Ignorar (Deadlock ignoring) . . . . .	2
1.2.1.2. Prevenir (Deadlock prevention) . . . . .	2
1.2.1.3. Evitar (Deadlock avoidance) . . . . .	2
1.2.1.4. Detectar y recuperar (Deadlock detection and recovery)	3
1.2.2. Revisión de la literatura . . . . .	3
1.2.2.1. Métodos de análisis estructural . . . . .	3
1.2.2.2. Enfoques basados en el grafo de alcanzabilidad . . . . .	4
1.3. Objetivos . . . . .	5
1.4. Requerimientos . . . . .	5
1.4.1. Listado de requerimientos . . . . .	6
1.5. Análisis de riesgos . . . . .	6
1.5.1. Listado de riesgos . . . . .	6
1.6. Estructura del texto . . . . .	7
<b>2. Marco teórico</b>	<b>9</b>
2.1. Redes de Petri . . . . .	9
2.1.1. Estructura de una red de Petri ordinaria . . . . .	10
2.1.2. Matriz de incidencia . . . . .	11
2.2. Dinámica de una red de Petri . . . . .	12
2.2.1. Disparo de una transición . . . . .	13
2.2.2. Función de transferencia y ecuación de estado . . . . .	14
2.2.3. Extensión de la ecuación de estado . . . . .	15
2.3. Propiedades de las redes de Petri . . . . .	16
2.3.1. Propiedades de limitación . . . . .	16
2.3.2. Propiedades de vivacidad . . . . .	16
2.3.3. Alcanzabilidad de una red de Petri . . . . .	17
2.3.4. Sifones y Trampas . . . . .	18
2.3.5. Invariantes de plazas y transiciones . . . . .	19
2.3.5.1. P-invariantes . . . . .	19
2.3.5.2. T-invariantes . . . . .	20
2.4. Concurrencia y sincronización . . . . .	21
2.4.1. Concurrencia y paralelismo . . . . .	21
2.4.1.1. Problemas inherentes a la programación concurrente .	22
2.4.1.2. Exclusión mutua . . . . .	22

2.4.2.	Interbloqueo . . . . .	23
2.4.2.1.	Condiciones para producir interbloqueo: . . . . .	23
2.4.3.	Sincronización . . . . .	24
2.4.3.1.	Semáforos . . . . .	24
2.4.3.2.	Monitores . . . . .	25
2.4.3.2.1.	Exclusión mutua en monitores . . . . .	25
2.4.3.2.2.	Condición de sincronización en monitores . . . . .	26
2.4.3.3.	Implementación de monitores con redes de Petri . . . . .	27
2.5.	S <sup>3</sup> PR . . . . .	28
2.5.1.	Definición de S <sup>2</sup> P . . . . .	28
2.5.2.	Definición de S <sup>2</sup> PR . . . . .	29
2.5.3.	Definición de S <sup>3</sup> PR . . . . .	31
<b>3.</b>	<b>Desarrollo</b>	<b>33</b>
3.1.	Desarrollo de la investigación . . . . .	33
3.2.	Modelo de desarrollo . . . . .	33
3.3.	Iteración 1: Algoritmo v1.0 . . . . .	34
3.3.1.	Introducción . . . . .	34
3.3.2.	Objetivos . . . . .	34
3.3.3.	Desarrollo . . . . .	35
3.3.3.1.	Herramienta a utilizar . . . . .	35
3.3.3.1.1.	Datos necesarios . . . . .	35
3.3.3.2.	Conversión de datos . . . . .	35
3.3.3.3.	Construcción del algoritmo . . . . .	36
3.3.4.	Conclusiones . . . . .	38
3.4.	Iteración 2: Algoritmo v2.0 . . . . .	38
3.4.1.	Introducción . . . . .	38
3.4.2.	Objetivos . . . . .	38
3.4.3.	Desarrollo . . . . .	38
3.4.4.	Conclusiones . . . . .	41
3.5.	Iteración 3: Algoritmo v3.0 . . . . .	42
3.5.1.	Introducción . . . . .	42
3.5.2.	Objetivos . . . . .	42
3.5.3.	Desarrollo . . . . .	42
3.5.3.1.	Definición del supervisor . . . . .	43
3.5.4.	Implementación del algoritmo . . . . .	46
3.5.4.1.	Desarrollo . . . . .	47
3.5.4.2.	Pseudocódigo . . . . .	49
3.5.5.	Criterio de elección del supervisor . . . . .	51
3.5.6.	Ejecución en diferentes escenarios . . . . .	51
3.5.6.1.	Caso Panamá . . . . .	52
3.5.6.1.1.	Características generales . . . . .	52
3.5.6.1.2.	Análisis estructural . . . . .	53
3.5.6.1.3.	Red controlada . . . . .	53
3.5.6.2.	Caso Ezpeleta . . . . .	54
3.5.6.2.1.	Características generales . . . . .	54
3.5.6.2.2.	Análisis estructural . . . . .	54
	Subred izquierda . . . . .	55

Subred derecha . . . . .	55
Control subred derecha . . . . .	56
Control red original . . . . .	56
3.5.6.3. Caso POPN . . . . .	57
3.5.6.3.1. Características generales . . . . .	58
3.5.6.3.2. Análisis estructural . . . . .	58
Subred derecha . . . . .	59
Subred izquierda . . . . .	59
Control de subredes . . . . .	60
3.5.6.3.3. Control red original . . . . .	61
3.5.6.4. Caso Guanjun . . . . .	62
3.5.6.4.1. Características generales . . . . .	62
Análisis estructural . . . . .	62
Red controlada . . . . .	63
3.5.7. Conclusión . . . . .	63
<b>4. Testing</b>	<b>65</b>
4.1. Nuevos escenarios . . . . .	65
4.1.1. Caso Hiuxia . . . . .	65
4.1.1.1. Características generales red Hiuxia . . . . .	65
4.1.1.2. Análisis estructural red Hiuxia . . . . .	65
4.1.1.2.1. Red controlada . . . . .	66
4.1.2. Caso JianChao . . . . .	67
4.1.2.1. Características generales . . . . .	67
4.1.2.2. Análisis estructural . . . . .	67
4.1.2.2.1. Red controlada . . . . .	68
4.1.2.3. Caso Zhao . . . . .	69
4.1.2.3.1. Características generales . . . . .	69
Análisis estructural . . . . .	70
Red controlada . . . . .	70
4.1.3. Caso Auto . . . . .	71
4.1.3.1. Características generales red Auto . . . . .	71
4.1.3.2. Análisis estructural Auto . . . . .	71
4.1.3.2.1. Control de la red . . . . .	72
Primera ejecución . . . . .	72
Segunda ejecución . . . . .	73
Tercera ejecución . . . . .	73
<b>5. Conclusión</b>	<b>75</b>
5.1. Trabajo a futuro . . . . .	76
<b>A. Tutorial de como utilizar el software <i>Petrinator</i></b>	<b>77</b>
A.1. Ejecución del software . . . . .	77
<b>B. Ejecución del algoritmo completa y detallada</b>	<b>83</b>
B.1. Ejecución del caso Panamá . . . . .	83
<b>Bibliografía</b>	<b>87</b>



# Índice de figuras

2.1.	Red de Petri marcada. . . . .	10
2.2.	Transiciones sensibilizadas de la red de Petri. . . . .	13
2.3.	Nuevo marcado de la red de Petri. . . . .	14
2.4.	Red de Petri 3 estados posibles. . . . .	17
2.5.	Grafo de alcanzabilidad. . . . .	18
2.6.	Sifón y trampa. . . . .	18
2.7.	P-invariantes de la red de Petri. . . . .	19
2.8.	Sección crítica. . . . .	22
2.9.	Sección crítica. . . . .	23
2.10.	Estructura interna de un monitor. . . . .	27
2.11.	Monitor. . . . .	28
2.12.	Red de Petri S <sup>2</sup> P. . . . .	29
2.13.	Redes de Petri S <sup>2</sup> PR. . . . .	30
(a).	N1 . . . . .	30
(b).	N2 . . . . .	30
2.14.	Composición de una red de Petri S <sup>3</sup> PR. . . . .	31
3.1.	RdP con un estado de deadlock. . . . .	36
3.2.	RdP Canal de Panamá. . . . .	37
3.3.	Estados de deadlock. . . . .	37
(a).	Estado número 82. . . . .	37
(b).	Estado número 484. . . . .	37
3.4.	RdP con un estado de deadlock. . . . .	38
3.5.	Análisis de la Figura 3.4. . . . .	39
(a).	Sifón . . . . .	39
(b).	Grafo de alcanzabilidad. . . . .	39
3.6.	Control sobre la RdP. . . . .	39
3.7.	Sifones RdP Panamá. . . . .	40
3.8.	Sifón izquierdo. . . . .	40
3.9.	Sifón derecho. . . . .	41
3.10.	Control RdP Panamá, deadlock <i>true</i> . . . . .	41
3.11.	Retroalimentación entre el proceso y el supervisor . . . . .	43
3.12.	RdP Ezpeleta. . . . .	44
3.13.	RdP Ezpeleta con sifón a controlar y sus plazas complemento. . . . .	45
3.14.	RdP Ezpeleta y sus T-invariantes. . . . .	45
3.15.	Colocación de un supervisor en RdP Ezpeleta. . . . .	46
3.16.	Ejecución del algoritmo v3.0. . . . .	48
3.17.	Ejecución del algoritmo v3.0. . . . .	51
3.18.	Modelo del Canal de Panamá. . . . .	52
3.19.	RdP Canal de Panamá y sus T-invariantes. . . . .	53

3.20. RdP Canal de Panamá controlada.	54
3.21. RdP Ezpeleta y sus T-invariantes.	54
3.22. Subred izquierda Ezpeleta.	55
3.23. Subred derecha Ezpeleta.	55
3.24. Subred derecha Ezpeleta controlada.	56
3.25. RdP Ezpeleta controlada.	57
3.26. Modelado de partes del sistema POPN.	57
3.27. Trayectorias de la producción de las piezas.	58
3.28. RdP POPN y sus T-invariantes.	58
3.29. Subred derecha POPN.	59
3.30. Subred izquierda POPN.	59
3.31. Subred derecha POPN controlada.	60
3.32. Subred izquierda POPN controlada.	61
3.33. RdP POPN controlada.	61
3.34. RdP Guanjun y sus T-invariantes.	62
3.35. RdP Guanjun controlada.	63
4.1. RdP Hiuxia y sus T-invariantes.	66
4.2. RdP Hiuxia controlada	67
4.3. RdP JianChao y sus T-invariantes.	68
4.4. RdP JianChao controlada	69
4.5. RdP Zhao y sus T-invariantes.	70
4.6. RdP Zhao controlada.	71
4.7. RdP Auto y sus T-invariantes.	71
4.8. RdP Auto controlada	72
4.9. Supervisor con marcado cero.	73
A.1. Software Petrinator	77
A.2. Abrir un archivo	78
A.3. Ventana de selección de archivo .pflow	78
A.4. Red cargada	78
A.5. Submenú de análisis	79
A.6. Clasificación y propiedades de la red	80
A.7. Exportar invariantes	80
A.8. Exportar matrices	81
A.9. Exportar grafo de alcanzabilidad	81
A.10. Exportar sifones y trampas	82
B.1. RdP Panamá	83
B.2. Ejecución del primer análisis de la red	84
B.3. Supervisor agregado	85
B.4. Segunda Iteración: análisis de la red con supervisores	85
B.5. Red resultante de agregar/eliminar arcos	86

**Nota:** Todas las figuras que no presenten una referencia son de autoría propia.

# Índice de cuadros

1.1.	Listado de requerimientos . . . . .	6
1.2.	Listado de riesgos . . . . .	6
1.3.	R1 - Recurso de hardware insuficiente . . . . .	6
1.4.	R2 - Ausencia de algoritmo . . . . .	6
1.5.	R3 - Algoritmo inadecuado . . . . .	7
1.6.	R4 - Herramienta inadecuada . . . . .	7
1.7.	R5 - Ausencia/escasez de datos . . . . .	7
3.1.	Supervisores: RdP Panamá . . . . .	53
3.2.	Supervisores: RdP Ezpeleta (R) . . . . .	56
3.3.	Supervisores: RdP POPN (R) . . . . .	60
3.4.	Supervisores: RdP POPN (L) . . . . .	60
3.5.	Supervisores: RdP Guanjun . . . . .	63
4.1.	Supervisores: RdP Hiuxia . . . . .	66
4.2.	Supervisores: RdP JianChao . . . . .	68
4.3.	Supervisores: RdP Zhao . . . . .	70
4.4.	Supervisores: RdP Auto - Primera ejecución . . . . .	72
4.5.	Supervisores: RdP Auto - Segunda ejecución . . . . .	73
4.6.	Supervisores: RdP Auto - Tercera ejecución . . . . .	73



# Lista de acrónimos

<b>AMS</b>	Automated Manufacturing Systems.
<b>FIFO</b>	First In First Out.
<b>FMS</b>	Flexible Manufacturing Systems
<b>LAC</b>	Laboratorio de Arquitectura de Computadoras
<b>RdP</b>	Red de Petri
<b>RAS</b>	Resource Allocation System
<b>S<sup>3</sup>PR</b>	System of Simple Sequential Processes with Resources
<b>WP</b>	Work Process



## Capítulo 1

# Introducción

### 1.1. Motivación e importancia del proyecto

Las motivaciones para el desarrollo de este trabajo pueden dividirse en dos grandes pilares. Por un lado aquellas relacionadas al proyecto en sí, entre las cuales puede destacarse la necesidad de realizar una investigación y desarrollo de un algoritmo capaz de solucionar los problemas de vivacidad de las redes de Petri(RdP), puntualmente las que modelan Sistemas de procesos secuenciales simples con recursos( $S^3PR$ ). Sumado a que el mismo podría formar parte de un proyecto más robusto que se está desarrollando en el Laboratorio de Arquitectura de Computadoras, y esto es algo emocionante.

Por otra parte, existen sin duda ciertas motivaciones de naturaleza académica. Entre ellas se puede mencionar la integración de los conocimientos adquiridos a lo largo de la carrera de grado, la contribución a la comunidad de investigadores e incluso la puesta en práctica de procedimientos estrictos de investigación, de desarrollo de software y de documentación que nos serán sin duda de gran valor durante nuestro futuro ejercicio como profesionales.

### 1.2. Estado del arte

En 1962, Petri inventó un enfoque teórico de la red para modelar y analizar los sistemas de comunicación en su tesis [20]. Este modelo se basó en los conceptos de funcionamiento asíncrono y concurrente de las partes de un sistema y en la comprensión de que las relaciones entre las partes podían representarse mediante una red. Se ha realizado una gran cantidad de investigaciones tanto sobre la naturaleza como sobre la aplicación de las redes de Petri, la cuál parece estar en expansión. Se ha demostrado que las redes de Petri son muy útiles en el modelado, análisis, simulación y control de sistemas concurrentes.

El flujo simultáneo de varios procesos en un sistema de asignación de recursos (RAS), que compiten por un conjunto finito de recursos, puede conducir a un punto muerto. Un interbloqueo (deadlock) ocurre cuando un conjunto de procesos se encuentra en un estado de “espera circular”<sup>1</sup>, donde cada proceso del conjunto está esperando que un recurso sea liberado por otro proceso del conjunto mientras ocupa un recurso que, a su vez, es necesario por uno de los otros procesos. La noción de deadlock parcial o total es frecuente y es preferible la validación antes de la implementación para reducir los riesgos [13].

---

<sup>1</sup>Definido en el Marco Teórico (Sección 2.4.2)

Los estados de deadlock son una situación bastante indeseable en un sistema de fabricación automatizado. Su ocurrencia a menudo deteriora la utilización de recursos y pueden conducir a resultados catastróficos en sistemas críticos para la seguridad. Las redes de Petri son una herramienta matemática importante para manejar problemas de interbloqueo en sistemas de asignación de recursos.

Un sistema de fabricación flexible (FMS) o un sistema de fabricación automatizado (AMS) son un conglomerado de máquinas herramienta controladas numéricamente por computadora, amortiguadores, accesorios, robots, vehículos guiados automatizados (AGV) y otros dispositivos de manejo de materiales. Por lo general, exhibe un alto grado de uso compartido de recursos para aumentar la flexibilidad. La existencia de recursos compartidos puede conducir a condiciones de espera circular. En tal sistema, una vez que ocurren los puntos muertos, persisten y no se resolverían sin la intervención de seres humanos u otro agente externo.

En diversos estudios realizados por distintos autores, en busca de reducir estos estados de deadlock, se parte de la premisa de la existencia de cuatro estrategias para manejarlos.

### 1.2.1. Estrategias de manejo de deadlock

#### 1.2.1.1. Ignorar (Deadlock ignoring)

Ignorar los estados de deadlock, que se conoce como el algoritmo de Ostrich<sup>2</sup>, se emplea en un sistema de asignación de recursos si la probabilidad de que se produzcan puntos muertos es mínima y la aplicación de otras estrategias de control de deadlock es técnicamente difícil. En un FMS o AMS, ignorar el estado deadlock es factible y razonable desde el punto de vista técnico y económico si el grado de intercambio de recursos es bajo.

#### 1.2.1.2. Prevenir (Deadlock prevention)

Se logra controlando la solicitud de recursos y garantizando que nunca se produzcan estados de deadlock. Los recursos se otorgan a los procesos de tal manera que una solicitud de un recurso nunca conduce a situaciones de deadlock. El objetivo es imponer limitaciones a la evolución de un sistema. En este caso, el cálculo se realiza offline de forma estática y una vez establecida una política de control, el sistema ya no puede alcanzar esos estados indeseables. Una ventaja importante de los algoritmos de prevención de interbloqueo es que no requieren ningún costo de tiempo de ejecución, ya que los problemas se resuelven en las etapas de diseño y planificación del sistema. La principal crítica es que tienden a ser demasiado conservadores, lo que reduce la utilización de recursos y la productividad del sistema.

#### 1.2.1.3. Evitar (Deadlock avoidance)

Para evitar los estados de deadlock se concede un recurso a un proceso solo si el estado resultante es seguro. Un estado se denomina seguro si existe al menos una secuencia de ejecución que permite que todos los procesos se ejecuten hasta su finalización. Para decidir si el próximo estado es seguro si se asigna un recurso

<sup>2</sup>Concepto informático para denominar el procedimiento de algunos sistemas operativos. Esta teoría, acuñada por A. S. Tanenbaum, señala que dichos sistemas, en lugar de enfrentar el problema de los bloqueos mutuos asumen que estos nunca ocurrirán.

a un proceso se debe realizar un seguimiento del estado del sistema global. Esto significa que son necesarios un gran almacenamiento y una amplia capacidad de comunicación.

#### 1.2.1.4. Detectar y recuperar (Deadlock detection and recovery)

Se otorgan recursos a un proceso sin ningún control. El estado de la asignación de recursos y las solicitudes se examinan periódicamente para determinar si un conjunto de procesos está bloqueado. Este examen se realiza mediante un algoritmo de detección de interbloqueo. Si se encuentra un interbloqueo, el sistema se recupera abortando uno o más procesos interbloqueados y entregandole los recursos liberados a otros procesos. En la práctica de fabricación, a menudo se necesitan operadores humanos para esta estrategia y, por lo tanto, puede resultar muy costoso.

### 1.2.2. Revisión de la literatura

Las políticas de prevención de deadlock se han logrado ampliamente y han dado lugar a una gran cantidad de resultados. En esta sección, se revisan las estrategias de **prevención** de deadlock mediante el uso de redes de Petri y se desarrollan en base a diferentes técnicas como el análisis estructural y el análisis del grafo de alcanzabilidad.

#### 1.2.2.1. Métodos de análisis estructural

Ezpeleta et al. [8] utilizó una clase de redes de Petri, las del tipo S<sup>3</sup>PR y propuso un algoritmo para la asignación de recursos en FMS. El algoritmo propuesto agregó nuevos lugares a la red para imponer ciertas restricciones que prohíben la presencia de sifones vacíos.

Los trabajos de Huang et al. [27] y Huang y et al. [7] presentan una nueva política de prevención de deadlock para la clase de redes de Petri, donde los estados de deadlock están relacionados con sifones sin marcar. Se agregan dos tipos de lugares de control al modelo original para un sistema de fabricación flexible llamado lugar de control ordinario y lugar de control ponderado para evitar que los sifones se desmarquen.

En Li y Wei [12], se introduce el concepto de sifones elementales para diseñar un supervisor de red de Petri que haga cumplir la vivacidad para el mismo modelo de red de Petri. Basado en sifones elementales y conceptos de P-invariantes en redes de Petri, Li y Wei introducen un algoritmo de prevención de interbloqueo para una clase específica de redes de Petri que pueden modelar adecuadamente varios FMS [12], los sifones en un modelo de red de Petri se clasifican en dependientes y sifones elementales, y se agregan lugares de control para todos los sifones elementales, de modo que los sifones están controlados de forma invariante.

Huang [6] propone una nueva metodología para sintetizar supervisores para la asignación de recursos en los FMS; se considera la clase de red Petri, a saber, S<sup>3</sup>PR, donde los puntos muertos están relacionados con sifones mínimos no marcados; todos los sifones mínimos deben controlarse agregando plazas de control. En este estudio, el número de plazas de control se reduce utilizando el concepto de sifón elemental.

Chen et al. [3], presenta un algoritmo de prevención de deadlock y el concepto de extracción por sifón se utiliza para calcular sifones no marcados para el modelo de red de Petri. Primero, un algoritmo de extracción de sifón obtiene un sifón no marcado máximo, divide los lugares en él y determina un sifón necesario de los lugares divididos; esto se lleva a cabo para todos los sifones sin marcar. Luego, el algoritmo diseña un monitor conveniente para marcar cada sifón necesario hasta que el modelo de red de Petri controlado esté activo.

Liu et al. [14] propuso una variedad de algoritmos de control de interbloqueo para AMS con recursos no confiables, los monitores y las subredes de recuperación están diseñadas para sifones mínimos estrictos (sifón vacío) y recursos no confiables, respectivamente, y se utilizan dos tipos de arcos que son normales e inhibidores para conectar monitores con subredes de recuperación. Para obtener un supervisor con complejidad estructural, los sifones elementales extraídos de todos los sifones mínimos estrictos están obviamente controlados.

### 1.2.2.2. Enfoques basados en el grafo de alcanzabilidad

Viswanadham et al. [26] presentó métodos de asignación de recursos estáticos para eliminar los puntos muertos; En este estudio, se utiliza un grafo de alcanzabilidad del modelo de red de Petri para llegar al método de asignación de recursos estático. Se implementa un algoritmo de prevención de interbloqueo para un sistema de fabricación de tamaño pequeño que consta de una máquina y un vehículo guiado automatizado (AGV). Los autores observaron que el algoritmo propuesto se puede aplicar de manera efectiva sólo para sistemas de fabricación de tamaño pequeño.

El trabajo de Uzam [23, 22] propone y mejora el método basado en una teoría de regiones para diseñar un supervisor de red de Petri óptimo. El tamaño del grafo de alcanzabilidad del modelo de red de Petri es un problema importante para aplicar la política de prevención de interbloqueos a un modelo de red de Petri muy grande. Por lo tanto, propusieron un algoritmo de reducción para simplificar modelos de redes de Petri muy grandes con el fin de facilitar los cálculos necesarios. Basado en la teoría de las regiones, Uzam y Zhou [25] desarrollaron una política iterativa de prevención de interbloqueos para FMS. En su estudio, el grafo de alcanzabilidad de un modelo de red de Petri se divide en dos partes: zona libre de bloqueo (zona activa, LZ) y zona de bloqueo (DZ). La zona viva se desarrolla ya que el componente máximo fuertemente conectado contiene la marca inicial. La zona de interbloqueo contiene marcas desde las que no se puede alcanzar la marca inicial. FBM está definido como una marca en la zona de interbloqueo. Los interbloqueos se pueden eliminar prohibiendo el disparo de las transiciones habilitadas en FBM. En su trabajo, el enfoque presentado tiene dos problemas. Primero, no se puede garantizar un supervisor óptimo en general, incluso si existe un supervisor óptimo. En segundo lugar, en cada iteración se requiere el cálculo del grafo de alcanzabilidad total para verificar si las marcas en la zona de interbloqueo son accesibles. Este enfoque es fácil de usar y simple si el grafo de alcanzabilidad de un sistema es pequeño pero no puede garantizar la optimización del comportamiento del supervisor. Los monitores redundantes en el supervisor de red de Petri pueden existir cuando el supervisor está diseñado mediante un enfoque de control de sifón iterativo. Por tanto, Uzam et al. [24] introdujo un enfoque para identificar y eliminar los monitores redundantes mediante el cálculo del gráfico de accesibilidad de un modelo de red de Petri controlado. Si la red de Petri controlada no pierde la vivacidad cuando se eliminan los

monitores redundantes, entonces los monitores redundantes se pueden eliminar del supervisor.

El objetivo de esta sección fue presentar una revisión de la literatura sobre los distintos enfoques planteados hasta el momento respecto a la prevención de los estados de deadlock. Un enfoque híbrido de control de interbloqueos se refiere a aquel que combina distintas de estas planificaciones para tratarlos. La motivación esencial de plantear una estrategia de este tipo es aprovechar las ventajas o formalismos de múltiples de estas, evitando sus desventajas. Esto fue puntualmente lo que motivó a no regirse por una única estrategia o estudio planteado.

### 1.3. Objetivos

Como se anticipó en la motivación, el objetivo final es el desarrollo de un algoritmo capaz de determinar las plazas y arcos necesarios a incorporar a la red original para lograr la ejecución de manera controlada, alcanzando la vivacidad de la misma. Además, generar código a partir del control de los estados de deadlock.

Al tratarse de un trabajo de investigación en el cual se fue evolucionando sobre distintas situaciones que se fueron encontrando a lo largo de su desarrollo, fueron apareciendo diferentes objetivos intermedios (mencionados en las diferentes iteraciones del mismo) mediante los que se logró alcanzar el objetivo principal antes mencionado que es obtener una red viva o libre de deadlock (interbloqueo).

Para esto se propone usar como soporte el software Petrinator, el mismo se utilizará para simular la redes y extraer la información necesaria de las mismas. Para que esta última pueda ser utilizada por el algoritmo se propone el siguiente objetivo secundario:

- Implementar una interfaz que procese la información extraída del software antes mencionado.

### 1.4. Requerimientos

En la ingeniería, los requerimientos se utilizan como datos de entrada en la etapa de diseño del producto. Establecen qué debe hacer el sistema, aunque no especifican la manera en que debe hacerlo. La fase de captura y registro de requisitos puede estar precedida por una fase de análisis conceptual del proyecto.

### 1.4.1. Listado de requerimientos

ID	Descripción
$R_1$	Se debe usar el software Petrinator para la construcción, análisis y extracción de los archivos necesarios de la red de Petri.
$R_2$	El algoritmo debe ser capaz de leer los archivos de extensión '.html' de los diferentes análisis exportados del Petrinator y llevar a cabo su posterior procesamiento para su utilización.
$R_3$	El algoritmo debe converger en redes de Petri del tipo S <sup>3</sup> PR.

CUADRO 1.1: Listado de requerimientos.

## 1.5. Análisis de riesgos

Un riesgo es un evento o condición incierta que, en caso de ocurrir, tendrá consecuencias negativas sobre al menos uno de los requerimientos del proyecto. Por esta razón es importante identificarlos con anticipación para mitigarlos.

### 1.5.1. Listado de riesgos

ID	Descripción
$R_1$	Recursos de hardware insuficientes.
$R_2$	Ausencia de algoritmo.
$R_3$	Algoritmo inadecuado.
$R_4$	Herramienta inadecuada.
$R_5$	Ausencia/escasez de datos.

CUADRO 1.2: Listado de riesgos.

R1 - Recurso de hardware insuficiente
<b>Condición:</b> Los recursos de hardware disponibles (computador básico) son inferiores a los necesarios.
<b>Consecuencia:</b> No permite analizar redes de Petri demasiado complejas, es decir, que presentan un gran numero de estados.
<b>Efecto:</b> Limitar la implementación de nuestro algoritmo a redes con una cantidad reducida de estados.

CUADRO 1.3: R1 - Recurso de hardware insuficiente.

R2 - Ausencia de algoritmo
<b>Condición:</b> Inexistencia del algoritmo.
<b>Consecuencia:</b> No lograr alcanzar la vivacidad de una red de Petri.
<b>Efecto:</b> Irrealizabilidad del proyecto.

CUADRO 1.4: R2 - Ausencia de algoritmo.

**R3 - Algoritmo inadecuado**

**Condición:** Encontrar un algoritmo pero que no converge en todas las redes del mismo tipo.

**Consecuencia:** Iteraciones infinitas del algoritmo.

**Efecto:** Incremento del tiempo dedicado en la investigación para la readaptación del algoritmo.

CUADRO 1.5: R3 - Algoritmo inadecuado.

**R4 - Herramienta inadecuada**

**Condición:** Elección incorrecta de la herramienta para implementar la red de Petri y/o utilización errónea.

**Consecuencia:** La herramienta no se adapta a nuestros datos ni a los requerimientos.

**Efecto:** Ineficiencia del modelo.

CUADRO 1.6: R4 - Herramienta inadecuada.

**R5 - Ausencia/escasez de datos**

**Condición:** Escasez de la obtención de datos.

**Consecuencia:** Imposibilidad de testear/desarrollar el algoritmo.

**Efecto:** Irrealizabilidad del proyecto.

CUADRO 1.7: R5 - Ausencia/escasez de datos.

## 1.6. Estructura del texto

Aquí se listan los distintos capítulos que conforman el proyecto, presentando una breve descripción de su contenido. El escrito está compuesto por 5 capítulos, los apéndices y la bibliografía.

- **Capítulo 1 - Introducción:** Se exponen en este capítulo los aspectos más significativos del proyecto, donde se incluye las motivaciones que llevaron a realizar el mismo junto con una revisión del estado del arte relacionado, el análisis de riesgos y requerimientos junto con los objetivos propuestos para el trabajo de fin de grado.
- **Capítulo 2 - Marco teórico:** Aquí se abordan los conceptos necesarios para comprender el enfoque del proyecto, además que los mismos dan fundamento a las posteriores implementaciones prácticas.
- **Capítulo 3 - Desarrollo:** En este capítulo se analizan todas las herramientas que permitieron la implementación del algoritmo desarrollado en este proyecto. Incluye el desarrollo en base a la investigación realizada a partir del estado del arte y los conceptos teóricos mencionados en el Capítulo 2. Como también el algoritmo en sus 3 versiones, cada una con sus respectivos objetivos, desarrollo y conclusiones.

- **Capítulo 4 - Testing:** Se exponen nuevos escenarios con el objetivo de verificar y validar el rendimiento del algoritmo desarrollado.
- **Capítulo 5 - Conclusión:** Se presenta en este capítulo las conclusiones obtenidas tras la realización del trabajo y posibles vías de trabajos futuros.
- **Apéndices:** En los apéndices se proporciona al lector dos tutoriales, uno que exemplifica como desplegar el entorno de trabajo y el otro es la ejecución de la versión final del algoritmo desarrollado en este proyecto.
- **Bibliografía:** En esta parte final del documento, se muestran todas las referencias que se han consultado para el desarrollo del proyecto.

## Capítulo 2

# Marco teórico

### 2.1. Redes de Petri

Una red de Petri es un modelo gráfico, formal y abstracto para la representación de sistemas distribuidos y el análisis del flujo de información. Este modelo facilita la comprensión sobre la estructura y el comportamiento dinámico y estático del sistema modelado. Las redes de Petri son de utilidad principalmente en el diseño de sistemas de *hardware* y *software* para especificación, simulación y diseño de diversos problemas de ingeniería, especialmente útiles para representar procesos concurrentes, así como procesos donde puedan existir restricciones en cuanto a la simultaneidad, la precedencia o la frecuencia de eventos concurrentes [2].

Las redes de Petri están fuertemente asociadas a la teoría de grafos, ya que las mismas pueden representarse como un grafo dirigido bipartito compuesto por cuatro elementos [29]:

- *Plazas*: representan los estados del sistema. Las plazas son variables de estado que pueden tomar valores enteros.
- *Tokens*: los tokens figuran como puntos negros dentro de las plazas. Éstos representan el valor específico de una condición o estado y generalmente se traducen a la presencia o ausencia de algún recurso del sistema.
- *Transiciones*: las transiciones representan el conjunto de sucesos cuya ocurrencia produce la modificación de los estados (y en consecuencia del estado global) del sistema.
- *Arcos*: los arcos indican las interconexiones entre las plazas y las transiciones, estableciendo el flujo de tokens que sigue el sentido de la flecha.

Una vez definidos sus componentes, se puede decir que una red de Petri es un grafo dirigido con dos tipos de nodos: plazas y transiciones. Estos nodos están vinculados por arcos, los cuales sólo pueden conectar una plaza con una transición o viceversa. Por otro lado, una red de Petri puede ser descrita mediante dos componentes:

1. Una estructura de red
2. Un marcado inicial

La estructura de red hace referencia a la red en sí, mientras que el marcado inicial sólo representa el estado inicial del sistema (denominado estado **idle**), es decir, sin que ninguna transición haya sido disparada. Un ejemplo simple de una red de Petri

marcada se muestra en la Figura 2.1. Éste será utilizado en las secciones siguientes para ilustrar operaciones y/o propiedades de las redes de Petri.

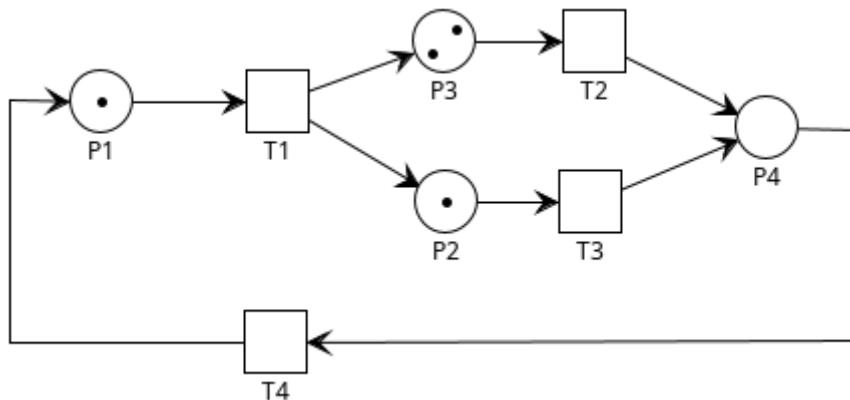


FIGURA 2.1: Red de Petri marcada.

### 2.1.1. Estructura de una red de Petri ordinaria

La estructura de una red de Petri puede definirse como una tupla de 5 elementos (5-tupla) [1] de la siguiente manera:

$$N = (P, T, I^-, I^+, M_0) \quad (2.1)$$

Donde:

- $P = \{P_1, P_2, \dots, P_n\}$  es un conjunto finito y no vacío que contiene todas las plazas de la red.
- $T = \{T_1, T_2, \dots, T_m\}$  es un conjunto finito y no vacío que contiene todas las transiciones.
- $I^-$  y  $I^+$  son las matrices *pre* y *post* respectivamente, cuya composición se abordará en la próxima sección.
- $M_0$  es el marcado inicial de la red. Definido como un vector con un elemento para cada plaza, donde  $M_0[i]$  contendrá la cantidad de *tokens* en la plaza  $i$  para el estado inicial.

Siguiendo con el ejemplo propuesto en la sección anterior (Figura 2.1), se puede representar la red como  $N = \{P, T, I^-, I^+, M_0\}$  donde:

- $P = \{P_1, P_2, P_3, P_4\}$
- $T = \{T_1, T_2, T_3, T_4\}$
- $M_0 = [1, 1, 2, 0]$

A continuación se explicará la manera de obtener las matrices  $I^-$  e  $I^+$ .

### 2.1.2. Matriz de incidencia

Las matrices  $I^-$  e  $I^+$  son las funciones de incidencia de entrada y salida de las plazas. Para el caso de la matriz  $I^+$ , denominada post, se tiene que cada elemento  $post(P_i, T_j)$  contiene el peso asociado al arco que va desde  $T_j$  hasta  $P_i$ . Este peso indica la cantidad de *tokens* que se generan en la plaza  $P_i$  cuando la transición  $T_j$  es disparada.

Por otro lado, en la matriz  $I^-$ , denominada pre, cada elemento  $pre(P_i, T_j)$  contiene el peso asociado al arco que va desde  $P_i$  hasta  $T_j$  e indica la cantidad de *tokens* que se retiran de la plaza  $P_i$  cuando se dispara la transición  $T_j$ .

Siguiendo con el ejemplo de la Figura 2.1, las matrices  $I^-$  e  $I^+$  asociadas son:

$$I^+ = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \end{pmatrix} \quad (2.2)$$

$$I^- = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (2.3)$$

Las filas de las matrices representan las plazas mientras que las columnas representan las transiciones, lo cual quiere decir que las matrices tendrán tantas filas como plazas tenga la red de Petri, y tantas columnas como transiciones.

De esta forma se puede observar como el elemento  $I^-[0][0]$  indica la relación de salida entre  $P_1$  y  $T_1$ . Más precisamente indica que cuando  $T_1$  se dispara, solo un *token* es retirado de  $P_1$  (ya que el peso del arco entre  $P_1$  y  $T_1$  es 1). De igual manera, el elemento  $I^+[0][0] = 0$  indica que cuando la misma transición se dispara, no se genera ningún token en  $P_1$  (ya que no existe ningún arco que parte de  $T_1$  hacia  $P_1$ ).

A partir de estas definiciones, se puede obtener la matriz de incidencia de la red. La misma está definida a continuación:

$$I = I^+ - I^- \quad (2.4)$$

Cabe aclarar que una red de Petri puede reconstruirse completamente a partir de sus matrices  $I^+$  e  $I^-$ , pero no así si se tiene sólo la matriz de incidencia  $I$ . Esto quiere decir que puede haber varias redes de Petri distintas con la misma matriz de incidencia, pero solamente una para las matrices  $I^+$  e  $I^-$ . Sin embargo, cuando una red de Petri no tiene autobuclees (presente cuando se tienen dos arcos (con sentidos contrarios) entre una misma plaza y transición), su matriz de incidencia determina completamente su estructura.

**EJEMPLO** La matriz de incidencia asociada a la red de Petri de la Figura 2.1 será entonces:

$$I = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \end{pmatrix} - \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} -1 & 0 & 0 & 1 \\ 1 & -1 & 0 & 0 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 1 & -1 \end{pmatrix} \quad (2.5)$$

## 2.2. Dinámica de una red de Petri

Se dice que una transición está sensibilizada cuando el marcado de todas las plazas entrantes a la transición es mayor o igual al peso de los arcos que las unen con dicha transición [17].

Antes de expresar la condición de sensibilizado de manera general, es necesario definir los siguientes conjuntos y funciones:

- $\bullet T_j$  es el conjunto compuesto por las plazas entrantes a  $T_j$ .
- $T_j \bullet$  es el conjunto compuesto por las plazas salientes de  $T_j$ .
- $\bullet \bullet P_i$  es el conjunto compuesto por las transiciones entrantes a las plazas que sensibilizan a las transiciones que le *agregan* tokens a  $P_i$ .
- $P_i \bullet \bullet$  es el conjunto compuesto por las transiciones entrantes a las plazas que sensibilizan a las transiciones que le *quitan* tokens a  $P_i$ .
- $m_k(P_i)$  es el marcado de la plaza  $P_i$  antes de disparar la transición  $T_j$ .
- $m_{k+1}(P_i)$  es el marcado de la plaza  $P_i$  después de disparar la transición  $T_j$ .
- $w_{ij}$  es el peso del arco  $P_i \rightarrow T_j$ .
- $w_{ji}$  es el peso del arco  $T_j \rightarrow P_i$ .

Entonces, el sensibilizado de una transición  $T_j$  está dado por:

$$T_j \text{ está sensibilizada si} \forall P_i \in \bullet T_j \Rightarrow m_k(P_i) > w_{ij} \quad (2.6)$$

Esta definición de sensibilizado es solo válida cuando los arcos que conectan las plazas con  $T_j$  son arcos comunes.

En la Figura 2.2 se resaltan las transiciones sensibilizadas del ejemplo anterior.

- $T_1, T_2, T_3$  están sensibilizadas, mientras  $T_4$  no está sensibilizada (ya que el marcado de  $P_4$  es menor al peso del arco  $P_4 \rightarrow T_4$ ).

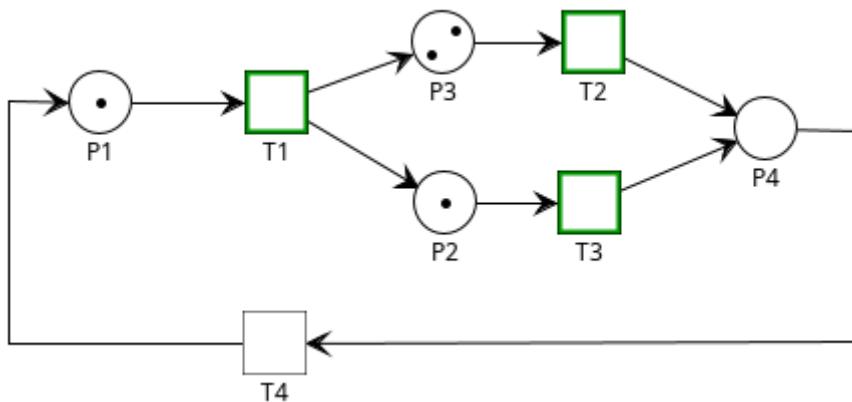


FIGURA 2.2: Transiciones sensibilizadas de la red de Petri.

### 2.2.1. Disparo de una transición

Si una transición está sensibilizada, la misma puede dispararse. El disparo de una transición resulta en un nuevo marcado de la red. Más precisamente, al ejecutarse una transición  $T_j$  con un marcado  $m_k$ , los marcados de las plazas pertenecientes a la red se alteran cumpliendo con las siguientes declaraciones:

$$\begin{aligned} \forall P_i \in \bullet T_j \Rightarrow m_{k+1}(P_i) &= m_k(P_i) - w_{ij} \\ \sigma(m_k, T_j) = \forall P_i \in T_j \bullet \Rightarrow m_{k+1}(P_i) &= m_k(P_i) + w_{ji} \\ \forall P_i \notin \bullet T_j \cup T_j \bullet \Rightarrow m_{k+1}(P_i) &= m_k(P_i) \end{aligned} \quad (2.7)$$

Es decir:

- Para todas las plazas entrantes a  $T_j$ , el nuevo marcado de cada plaza se habrá **decrementado** tantos *tokens* como peso tenga el arco  $P_i \rightarrow T_j$ .
- Para todas las plazas salientes de  $T_j$ , el nuevo marcado de cada plaza se habrá **incrementado** tantos *tokens* como peso tenga el arco  $T_j \rightarrow P_i$ .
- Para el resto de las plazas, el nuevo marcado será exactamente igual al que tenían antes del disparo de  $T_j$ .

Continuando con el ejemplo anterior, se puede observar en la Figura 2.3 el nuevo marcado de la red luego del disparo de la transición  $T_3$ :

- La única plaza entrante a  $T_3(P_3)$ , ha **decrementado** su marcado en 1 *token* (ya que el peso del arco  $P_3 \rightarrow T_3$  es 1).
- La plaza saliente de  $T_3(P_4)$  ha **incrementado** su marcado de acuerdo a los pesos de los arcos correspondientes, en este caso en 1.
- Las plazas que no es entrante ni saliente de  $T_3(P_3)$  ha mantenido su marcado original.

Cabe destacar que como consecuencia del disparo de  $T_3$ , se ha producido la sensibilización de  $T_4$ .

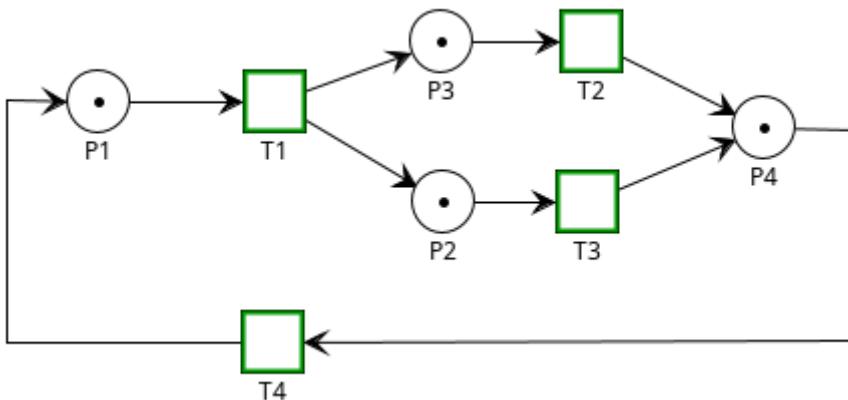


FIGURA 2.3: Nuevo marcado de la red de Petri.

## 2.2.2. Función de transferencia y ecuación de estado

Una vez explicada la dinámica del disparo de una transición y la forma de obtener la matriz de incidencia, se detalla una expresión matemática necesaria para obtener el nuevo marcado luego del disparo de una transición. La misma se denomina **función de transferencia** y está definida como el producto de la matriz de incidencia  $I$  con un vector  $\vec{\delta}$  cuyos componentes son todos ceros, exceptuando el componente asociado a la transición que se quiere disparar, cuyo valor será uno. Entonces, se tendrá:

$$I \cdot \vec{\delta} \quad (2.8)$$

donde, para el disparo de una transición  $T_j$ , se tiene:

- $\delta[j] = 1$
- $\delta[i] = 0 \forall i / i \neq j$

Por otro lado, es necesario introducir la **ecuación de estado** de las redes de Petri. Con esta ecuación es posible obtener el siguiente estado del sistema luego del disparo de una transición. Esta es una manera más simple que la metodología gráfica para analizar la evolución de los sistemas. La ecuación de estado en un tiempo  $i$ , para calcular el nuevo marcado de la red en un tiempo  $i + 1$  se define como:

$$M_{i+1} = M_i + I \cdot \vec{\delta} \quad (2.9)$$

donde  $M_{i+1}$  es el marcado luego del disparo de la transición,  $M_i$  es el marcado antes del disparo y el segundo término de la ecuación es la función de transferencia.

Siguiendo con el ejemplo hasta ahora analizado se verá cómo calcular el marcado de la red luego del disparo de la transición  $T_3$  haciendo uso de la ecuación de estado.

En la Figura 2.2 se observan las transiciones sensibilizadas de la red para el marcado inicial  $M_0$ .

La ecuación de estado requiere tres elementos:

1. El marcado antes del disparo. Éste es:

$$M_i = M_0 = \begin{pmatrix} 1 \\ 1 \\ 2 \\ 0 \end{pmatrix} \quad (2.10)$$

2. La matriz de incidencia de la red. La misma fue calculada en la Sección 2.1.2 y es la siguiente:

$$I = \begin{pmatrix} -1 & 0 & 0 & 1 \\ 1 & -1 & 0 & 0 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 1 & -1 \end{pmatrix} \quad (2.11)$$

3. El vector de disparo  $\vec{\delta}$ , que tendrá tantos elementos como transiciones haya en la red, cuyos valores serán cero para todas las transiciones excepto para aquella que se desee dispara:

$$\vec{\delta} = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} \quad (2.12)$$

con lo cual el nuevo marcado está definido por:

$$M_{i+1} = \begin{pmatrix} 1 \\ 1 \\ 2 \\ 0 \end{pmatrix} + \begin{pmatrix} -1 & 0 & 0 & 1 \\ 1 & -1 & 0 & 0 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 1 & -1 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} \quad (2.13)$$

resultado que coincide con lo obtenido en la Figura 2.3. La ecuación de estado representa matemáticamente el comportamiento dinámico del sistema, permitiendo calcular el nuevo estado del mismo luego de la ocurrencia de un evento a través de una simple ecuación.

### 2.2.3. Extensión de la ecuación de estado

Como se mencionó en la sección anterior, la ecuación 2.9 permite calcular el siguiente estado luego del disparo de una transición. Sin embargo, puede que se deseé obtener el marcado final luego de una secuencia de disparos. Suponiendo que se parte del estado inicial  $M_0$ , esto puede representarse como:

$$M_i = M_0 + I \cdot \sum_{j=1}^i u_j \quad (2.14)$$

donde, la sumatoria representa un vector asociado a la secuencia de transiciones que se desea disparar y se denomina vector  $S$ . Para ejemplificar, el cálculo de un marcado  $M_i$  a partir del marcado inicial y luego del disparo de las transiciones  $T_3, T_4, T_1, T_2$ , está dado por:

$$M_i = M_0 + I \cdot \vec{S} \quad (2.15)$$

donde  $\vec{S} = \{ 1, 1, 1, 1 \}$  e  $I$  es la matriz de incidencia asociada a la red.

## 2.3. Propiedades de las redes de Petri

### 2.3.1. Propiedades de limitación

Dada una red de Petri definida por  $PN = \{P, T, I^-, I^+, M_0\}$ , se dice que una plaza  $P$  está  $k$ -limitada si existe un número entero  $k$  que, para todo marcado posible de la red, se verifica que la cantidad de tokens de la plaza siempre es igual o menor a  $k$ . Es decir:

$$\exists k \in N / \forall M \in \text{marcados}(PN) \Rightarrow M(P) \leq k \quad (2.16)$$

Por otro lado, se dice que la red está  **$k$ -limitada** si todas las plazas que contiene son  **$k$ -limitadas**.

A partir de la definición de limitación surgen varios conceptos, entre los cuales se encuentran los siguientes:

- Una red de Petri es **segura** si todas sus plazas son **1-limitadas**. Esto significa que nunca puede darse un disparo si la plaza de llegada ya contiene un *token*.
- Una red de Petri es **cíclica** si siempre existe la posibilidad de alcanzar el marcado inicial desde cualquier otro marcado alcanzable. Es decir,  $\forall M \in \text{marcados}(PN), M_0$  es dinámicamente alcanzable desde  $M$ .
- Una red de Petri es **repetitiva** si existe una secuencia de disparos  $\sigma$  que contiene todas las transiciones de la red y existe un marcado  $M$  que para el cual  $M \xrightarrow{\sigma} M$ . Es decir, existe una secuencia de disparos que contiene todas las transiciones y que lleva la red del marcado actual al mismo marcado.
- Una red de Petri es **conservativa** si se cumple que  $\forall M \in \text{marcados}(PN)$ , el número total de *tokens* en el marcado  $M$  es igual al número de tokens en el marcado  $M_0$ . En otras palabras, la red siempre contiene la misma cantidad de marcas.

### 2.3.2. Propiedades de vivacidad

La **vivacidad** de una transición indica que, en todo instante de la evolución de la red, su disparo es posible. Este concepto es particularmente relevante ya que determina si la ejecución de la red puede o no detenerse en un estado determinado. A partir de esto se puede definir la vivacidad de una red de Petri. Esta propiedad indica que una red  $N = \{P, T, I^-, I^+, M_0\}$  es viva para un marcado si todas sus transiciones lo son.

Por otro lado, la **cuasi-vivacidad** de una transición expresa la posibilidad de dispararla al menos una vez a partir de un marcado inicial  $M_0$ . De la misma manera que para el caso de la vivacidad, una red de Petri es cuasi-viva si todas sus transiciones lo son.

Gracias a esta última definición, se puede definir la vivacidad en función de la quasi-vivacidad de la siguiente manera: una transición es viva si la misma es quasi-viva en la red para todo marcado alcanzable desde  $M_0$ .

La vivacidad está directamente asociada con la ausencia de **deadlock** o interbloqueo. En términos generales, el deadlock es el bloqueo permanente de un conjunto de procesos o hilos de ejecución en un sistema concurrente que compiten por recursos del sistema o bien se comunican entre ellos. En el caso de una red de Petri, esto suele ocurrir cuando dos o más transiciones esperan mutuamente por el disparo de la otra, produciendo el bloqueo permanente de esa porción de la red. Una red de Petri viva garantiza la ausencia de interbloqueo sin importar la secuencia de disparos.

### 2.3.3. Alcanzabilidad de una red de Petri

La **alcanzabilidad** de una red de Petri es fundamental para el análisis de las propiedades dinámicas de un sistema. A grandes rasgos, permite determinar si el sistema modelado puede alcanzar un determinado estado.

Un marcado  $M_i$  es alcanzable desde  $M_0$  si existe una secuencia finita de disparos  $\sigma$  tal que  $M_0 \xrightarrow{\sigma} M_i$ .

Los marcados alcanzables por la red pueden ser representados como nodos de un grafo o árbol, donde los arcos indican los disparos necesarios para alcanzar dicho marcado. El algoritmo para determinar el árbol de alcanzabilidad de una red de Petri será explicado con detalle en el desarrollo del proyecto.

Entonces, el grafo de alcanzabilidad  $A$  se define como el menor conjunto que cumpla con las expresiones 2.17 y 2.18.

$$M_0 \in A \quad (2.17)$$

Esta condición simplemente aclara que el marcado inicial de la red siempre forma parte del grafo de alcanzabilidad, ya que el mismo no requiere ninguna secuencia de disparos para ser alcanzable.

$$\forall M \text{ sii } M \xrightarrow{\sigma} M_i \Rightarrow M_i \in A \quad (2.18)$$

Esto significa que para cualquier marcado  $M$ , si a partir del mismo puede alcanzarse otro marcado  $M_i$ , entonces  $M_i$  forma parte del grafo.

**EJEMPLO** Suponiendo una red simple como la de la Figura 2.4, compuesta por tres plazas y una única transición  $T_1$ , se puede afirmar que sólo hay tres estados posibles en la red:

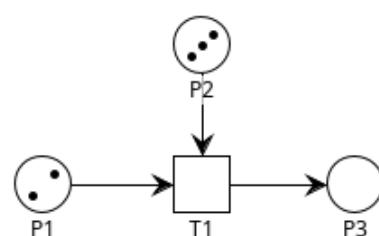


FIGURA 2.4: Red de Petri 3 estados posibles.

- El marcado inicial  $[2, 3, 0]$

Luego del segundo disparo no existen disparos posibles. Por lo tanto, la red de la Figura 2.4 produce el grafo de alcanzabilidad mostrado en la Figura 2.5.

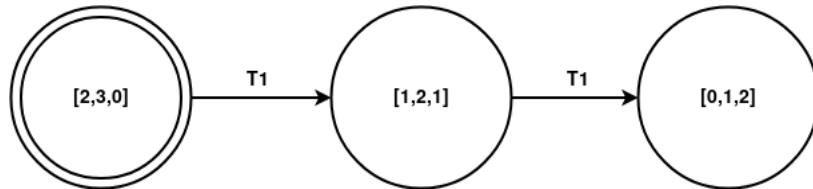


FIGURA 2.5: Grafo de alcanzabilidad.

### 2.3.4. Sifones y Trampas

Los conceptos de sifón y trampa están directamente relacionados con las propiedades de **interbloqueo** y **vivacidad** de una red de Petri.

Un **sifón** se define como un subconjunto no vacío de plazas  $S$  para el cual se cumple que el subconjunto de transiciones entrantes a  $S$  está contenido dentro del subconjunto de transiciones salientes de  $S$ . En otras palabras, un grupo de plazas es un sifón si, una vez que un token sale del grupo de dichas plazas, el mismo nunca puede volver a entrar. Se dice que un sifón  $S$  es **mínimo** si no contiene otro sifón como un subconjunto. En un sifón mínimo debe existir al menos dos lugares; de lo contrario, la estructura restante no puede considerarse un sifón. En la Figura 2.6 se puede observar una red que contiene una trampa y un sifón.

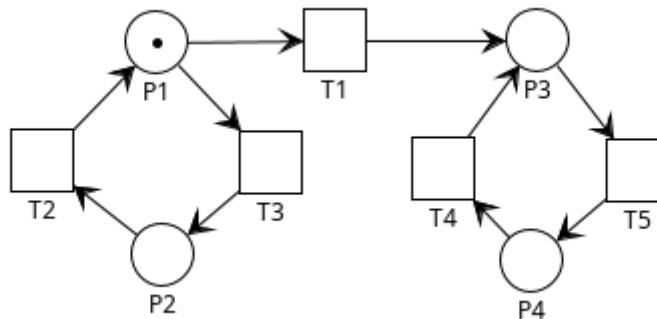


FIGURA 2.6: Sifón y trampa.

Tomando el subconjunto de plazas  $S = \{P_1, P_2\}$  se deben obtener los siguientes subconjuntos de transiciones:

- El subconjunto  $\bullet S = \{T_2, T_3\}$  será aquel compuesto por las transiciones entrantes a las plazas que componen  $S$ .
- El subconjunto  $S\bullet = \{T_2, T_3\}$  será aquel compuesto por las transiciones entrantes a las plazas que componen  $S$ .

Por propiedad de los sifones, para que  $S$  pueda considerarse como tal debe cumplirse que:

$$\bullet S \subseteq S\bullet \quad (2.19)$$

En este caso, se comprueba que  $T_2, T_3 \subseteq \{T_1, T_2, T_3\}$ , quedando demostrado que  $\{P_1, P_2\}$  es en efecto un sifón y que, si la transición  $T_1$  se dispara, el token removido de  $P_1$  nunca volverá a ingresar al subconjunto.

Por otro lado, una **trampa** se define como un subconjunto de plazas  $G$  para el cual se cumple que el subconjunto de transiciones salientes de  $G$  está contenido dentro del subconjunto de transiciones entrantes a  $G$ . Esto quiere decir que un conjunto de plazas constituyen una trampa si una vez que un token entra dicho grupo éste nunca vuelve a salir.

Siguiendo con el ejemplo de la Figura 2.6, se analizará el subconjunto de plazas  $G = \{P_3, P_4\}$ . Los subconjuntos de transiciones serán:

- El subconjunto  $\bullet G = \{T_1, T_4, T_5\}$  será aquél compuesto por las transiciones entrantes a las plazas que componen  $G$ .
- El subconjunto  $G\bullet = \{T_4, T_5\}$  será aquél compuesto por las transiciones salientes de las plazas que componen  $G$ .

Por propiedad de las trampas, para que  $G$  pueda considerarse como tal, debe cumplirse que:

$$G\bullet \subseteq \bullet G \quad (2.20)$$

Propiedad que es simplemente comprobable ya que  $T_4, T_5 \subseteq \{T_1, T_4, T_5\}$ , demostrando que  $\{P_3, P_4\}$  es una trampa.

### 2.3.5. Invariantes de plazas y transiciones

Las invariantes de una red son propiedades independientes tanto del marcado inicial como de la secuencia de disparos, y pueden asociarse a ciertos subconjuntos de plazas o de transiciones; con lo cual surgen dos conceptos:

#### 2.3.5.1. P-invariantes

Una **invariante de plazas** o **P-invariante** es un conjunto de plazas cuya suma de tokens no se modifica con una secuencia de disparos arbitraria. Esto se puede observar en el ejemplo de la Figura 2.7:

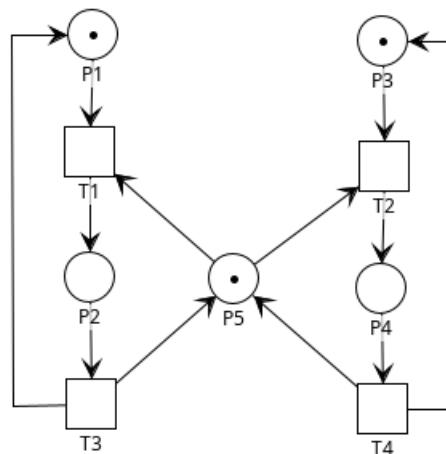


FIGURA 2.7: P-invariantes de la red de Petri.

Tras el análisis de la red , se obtienen las siguientes invariantes de plazas:

$$\begin{aligned} m(P_1) + m(P_2) &= 1 \\ m(P_3) + m(P_4) &= 1 \\ m(P_2) + m(P_4) + m(P_5) &= 1 \end{aligned} \quad (2.21)$$

El primer ítem expresa que la sumatoria de tokens en las plazas  $P_1$  y  $P_2$  siempre será igual a uno, afirmación completamente observable al mirar la red de la Figura 2.7. Estas declaraciones implican la siguiente consecuencia:

$$I.x = 0 \quad (2.22)$$

donde  $I$  es la matriz de incidencia y  $x$  es un vector característico de un subconjunto  $Q$  de las plazas que forman parte de la invariante (un uno en una posición indica que esa plaza es parte de la invariante y un cero indica lo contrario). A partir de esto surge la siguiente fórmula:

$$\sum_{P \in \bullet t \cap Q} W(p, t) = \sum_{P \in t \bullet \cap Q} W(t, p) \quad (2.23)$$

la cual puede ser expresada en función de un vector  $t$  de la siguiente manera:

$$\sum_{P \in \bullet t \cap Q} t(p) = \sum_{P \in t \bullet \cap Q} t(p) \quad (2.24)$$

Esto quiere decir que:

$$\sum_{P \in (\bullet t \cup t) \cap Q} t(p) = 0 \quad y \quad \sum_{P \in Q} t(p) = 0 \quad (2.25)$$

Si se remplaza  $Q$  por los vectores característicos  $I_q$  , estas dos igualdades pueden escribirse como:

$$\sum_{P \in Q} t(p) I_q(p) = 0 \quad y \quad \sum_{p \in P} t(p) I(p) = 0 \quad (2.26)$$

Lo cual es simplemente la definición del producto escalar entre dos vectores:

$$t.I_q = 0 \quad (2.27)$$

Como los disparos son arbitrarios, se puede establecer la siguiente relación:

$$t_j I_q; \forall t_j \in T \iff I^T I_q = 0 \quad (2.28)$$

donde  $I^T$  es la matriz de incidencia transpuesta.

### 2.3.5.2. T-invariantes

Un **invariante de transición** ó **T-invariante** es el conjunto de transiciones que deben dispararse para que la red de Petri retorne a su estado inicial.

Como se mencionó en el apartado anterior, para el cálculo de las P-invariantes se hace uso de la ecuación  $I.x = 0$ , siendo  $I$  la matriz de incidencia y  $x$  un vector característico constituido por las plazas que forman parte de la invariante. En este caso, para el cálculo de los vectores que constituyen las T-invariantes, la ecuación

asociada será similar a las P-invariantes, a diferencia que se hace uso de  $I^T$  en vez de  $I$ :

$$I^T \cdot x = 0 \quad (2.29)$$

Aquí, a diferencia de las P-invariantes, el vector  $x$  está constituido por el conjunto de transiciones que deben dispararse para que la red retorne al estado inicial. Un “1” en una posición indica que esa transición es parte de la invariante y un “0” indica lo contrario.

Tomando la Figura 2.7 como ejemplo y se obtienen las siguientes invariantes de transiciones:

$$T - \text{invariantes} = \begin{pmatrix} T0 & T1 & T2 & T3 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix} \quad (2.30)$$

Ambos vectores cumplen la condición planteada ( $I^T \cdot x = 0$ ) y si se observa la imagen en cuestión, se puede apreciar que la red retorna a su estado inicial si las transiciones especificadas en los vectores se disparan.

## 2.4. Concurrencia y sincronización

En los sistemas de computación actuales conviven múltiples procesos que cooperan para lograr determinados objetivos y compiten por recursos del sistema, entre ellos el procesador, la memoria RAM, los puertos de entrada/salida, etc.

Dado que generalmente el numero de procesos de un sistema supera ampliamente el numero de recursos, se deben establecer formas de comunicación y sincronización entre ellos que hagan que el sistema funcione correctamente.

En ésta sección se definirá cuando dos programas son concurrentes y/o paralelos y las condiciones que deben cumplirse para que dos secciones de código fuente puedan ser ejecutadas de manera concurrente. Luego, se verá que la ejecución concurrente de procesos trae aparejados ciertos problemas como el interbloqueo y la inanición.

Por esta razón, deben ejecutarse ciertos mecanismos de control para garantizar la correcta ejecución de los programas, entre ellos, los semáforos y monitores.

El objetivo de esta sección es que el lector adquiera una idea general sobre la programación concurrente y sobre los problemas inherentes a la misma.

### 2.4.1. Concurrencia y paralelismo

Dos procesos serán concurrentes cuando la primera instrucción de uno de ellos se ejecuta después de la primera instrucción de otro proceso y antes de la última. No es necesario que estos se ejecuten al mismo tiempo, basta con el hecho de que se intercalen sus instrucciones. En caso de ejecutarse al mismo tiempo se dice que hay programación paralela. La programación concurrente es un paralelismo potencial, dependiente del hardware que lo soporte [19].

#### 2.4.1.1. Problemas inherentes a la programación concurrente

La intercalación de instrucciones de diferentes procesos, debe ser bien manejada y controlada dado que puede producir mal funcionamiento del sistema. Los problemas inherentes a la concurrencia son:

- **Exclusión mutua:** se debe garantizar que si un proceso adquiere el recurso los demás deberán esperar hasta que sea liberado.
- **Condición de sincronización:** hay situaciones en las que un proceso debe esperar que ocurra algún determinado evento para poder continuar. Por ello se debe garantizar que si el evento **no** ocurrió, el proceso **no** continúe.
- **Interbloqueo:** esta situación se produce cuando todos los procesos están esperando un evento que nunca ocurrirá. Se debe garantizar que estas situaciones no ocurran.
- **Inanición:** en este caso, el sistema en su conjunto hace progresos, pero existe un grupo de procesos que nunca progresaran pues no se les otorga tiempo de procesador para hacerlo.

#### 2.4.1.2. Exclusión mutua

La exclusión mutua implica que dos o más procesos intentan acceder a un único recurso compartido entre ellos pero solo uno puede acceder a cada instante. Cuando se da un caso de estas características, se desea que todo lo que necesite hacer unos de los procesos sobre el recurso se realice de manera indivisible y luego lo deje disponible para que otro proceso ejecute sus instrucciones sobre el recurso.

A la porción de código que se desea que se ejecute de manera indivisible o atómica se le llama **sección crítica**. Se debe lograr que todas las instrucciones dentro de la sección crítica se ejecuten en exclusión mutua lo que implica que el hecho de que cuando uno de los procesos este ejecutando esa porción de código el resto no podrá hacerlo.

**Solo uno de los procesos podrá estar en la sección crítica en un instante dado.**

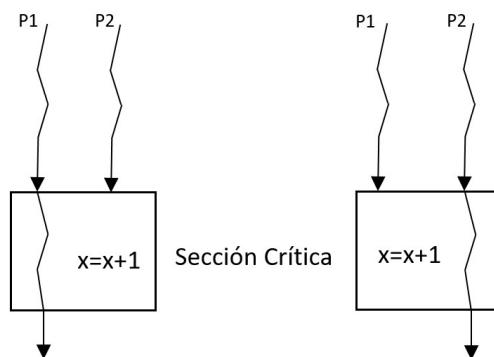


FIGURA 2.8: Sección crítica.

En la Figura 2.8 se observa como dos procesos  $P_1$  y  $P_2$  intentan ejecutar una porción de código de una sección crítica. La imagen de la izquierda (a) muestra que

el proceso  $P_1$  consigue ingresar a ejecutar la sección crítica. La imagen de la derecha (b) muestra que el proceso  $P_2$  puede ingresar solo cuando el proceso  $P_1$  ya no está en la misma.

La exclusión mutua se puede representar de la siguiente manera.

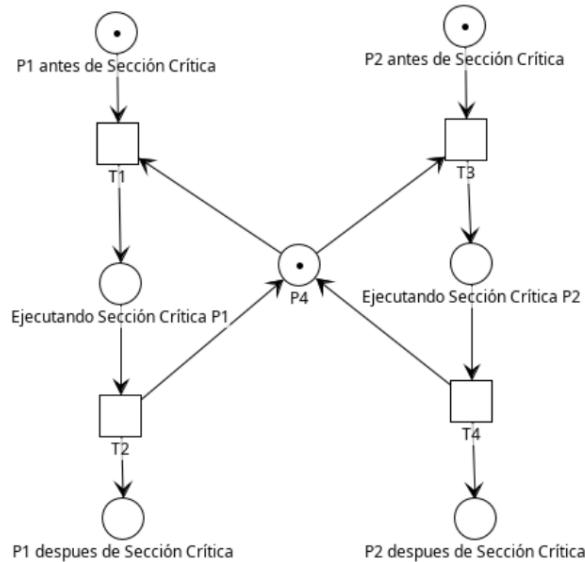


FIGURA 2.9: Sección crítica.

En la Figura 2.9 la plaza  $MUTEX$  esta limitada a un único token y el análisis de invariantes de plazas demuestra formalmente la propiedad de exclusión mutua entre los procesos  $P_1$  y  $P_2$ .

$$m(EjecutandoSCP1) + m(MUTEX) + m(EjecutandoSCP2) = 1 \quad (2.31)$$

## 2.4.2. Interbloqueo

En un sistema donde los procesos compiten por limitados recursos, pueden producirse demandas contradictorias de los mismos. Por ejemplo, si existen dos procesos,  $A$  y  $B$ , y dos recursos  $R_1$  y  $R_2$ , y ambos procesos necesitan los dos recursos para proseguir, si el proceso  $A$  toma el recurso  $R_1$  y el  $B$  el recurso  $R_2$ , ambos procesos se bloquearan a la espera del otro recurso, pero ninguno liberará el recurso que posee hasta no conseguir los dos. A esta situación se la conoce como interbloqueo [21].

### 2.4.2.1. Condiciones para producir interbloqueo:

Deben presentarse tres condiciones de gestión para que sea posible un interbloqueo:

1. *Exclusión mutua*: sólo un proceso puede utilizar un recurso en cada momento. Ningún proceso puede acceder a un recurso que se ha asignado a otro proceso.
2. *Retención y espera*: un proceso puede mantener los recursos asignados mientras espera la asignación de otros recursos.
3. *No apropiación*: ningón proceso podrá ser forzado a abandonar un recurso que retiene.

4. *Espera circular*: existe una cadena cerrada de procesos donde cada proceso tiene un recurso que necesita un proceso que le sigue en la cadena.

Las tres primeras condiciones son necesarias pero no suficientes para que exista interbloqueo. La cuarta es una consecuencia potencial de las tres primeras y, en caso de darse, generará una **espera circular irresoluble**. Esta es de hecho la definición de interbloqueo.

### 2.4.3. Sincronización

Para solucionar los problemas inherentes a la programación concurrente, se utiliza lo que se llama *sincronización entre los procesos*.

Se habla de sincronización, en general, cuando determinados fenómenos ocurren o deben ocurrir en un determinado orden o a la vez.

Para la computación, la sincronización es representada por las señales que se envían los procesos para colaborar entre ellos o para indicar el estado de recursos compartidos, para indicar que un evento o acción ocurrió o no y determinar la continuidad o no de un proceso, etcétera.

La condición de sincronización puede definirse como la propiedad requerida para que un proceso no realice ninguna acción o evento hasta que otro proceso realice una determinada acción o evento.

#### 2.4.3.1. Semáforos

Los semáforos son un sistema de señales simples utilizadas por los procesos para comunicarse entre ellos y lograr la sincronización requerida. Estos tienen una variable de sincronización, del tipo entero no negativo, que indica la cantidad de recursos disponibles. Sobre esta se realizan dos tipos de operaciones:

- *wait*: decremente el valor del semáforo solo si este es mayor que cero. Este proceso indica que se utiliza uno de los recursos que controla el semáforo. Si el valor del semáforo al momento de ejecutar la operación *wait* es cero, indica que no hay recursos disponibles y el proceso deben bloquearse hasta que se libere alguno.
- *signal*: es la acción de liberar un recurso que estaba siendo utilizado. En caso de haber algún proceso bloqueado en el semáforo se lo despierta para que utilice el recurso. De no existir algún proceso, se incrementa el valor del semáforo.

Los semáforos son primitivas con las cuales es difícil expresar una solución a grandes problemas de concurrencia, ya que tienen algunas debilidades:

- La omisión de una de las primitivas puede corromper el funcionamiento de un sistema concurrente.
- El control de concurrencia es responsabilidad del programador.
- Las primitivas de control se encuentran esparcidas por todo el código, lo que hace muy difícil la corrección de errores y el mantenimiento del mismo.

Debido a estas razones existe otro mecanismo de software para el control de concurrencia denominado **monitor**.

### 2.4.3.2. Monitores

Como se dijo, los semáforos, generalmente se encuentran dispersos en el código, lo que lo hace más confuso y muchas veces es difícil notar cual es el recurso compartido y determinar si está correctamente sincronizado. Por ello, se necesita un sistema que sea igual de versátil que los semáforos pero que permita efectuar un control más estructurado de la exclusión mutua. Una herramienta con estas características fue propuesta por C.A.R Hoare en 1975 y es conocida como *monitor*.

Un **monitor** es un mecanismo de abstracción de datos, lo que permite representar de forma abstracta un recurso compartido mediante variables que indican su estado. El acceso a esas variables solo es posible a través de un conjunto de funciones/métodos que el monitor exporta al exterior.

Un monitor se compone de los siguientes elementos:

- Un *conjunto de variables* locales que pueden denominarse permanentes. Se utilizan para almacenar el estado interno del recurso que representa el monitor. Se denominan permanentes ya que permanecen sin modificarse entre dos llamadas consecutivas al monitor y solo pueden ser accedidas dentro del mismo.
- Un *código de inicialización* que se ejecuta antes que la primera instrucción ejecutable del programa y su fin es inicializar las variables permanentes.
- Un *conjunto de procedimientos internos* que manipulan las variables permanentes.
- Una *declaración de los procedimientos* que son *exportados* y pueden ser accedidos por los procesos activos externos.

#### 2.4.3.2.1. Exclusión mutua en monitores

El control de la exclusión mutua en un monitor se basa en la existencia de una cola asociada al mismo que se denominara *cola del monitor*. La gestión de esta cola se realiza de la siguiente manera:

1. Cuando un proceso activo está dentro del monitor (ejecutando alguno de los procedimientos del mismo) y aparece otro proceso activo que intenta ejecutar otro (o el mismo) procedimiento, el código de acceso al monitor bloquea el proceso que realiza la llamada y lo inserta en la cola del monitor (con política FIFO). Así, se impide que dos procesos estén al mismo tiempo dentro del monitor.
2. Cuando un proceso activo abandona el monitor, este último selecciona el proceso que está al frente de la cola del monitor y lo desbloquea para que comience a ejecutar las operaciones que le solicitó al monitor. Si la cola estaba vacía, el monitor queda libre y cualquier proceso activo que llame alguno de sus procedimientos entrará al monitor.

Esto asegura que las variables compartidas nunca son accedidas concurrentemente. Una cuestión importante es que la responsabilidad de bloquear un proceso es del monitor y no del proceso. Al comparar este sistema con un semáforo se ve que en el caso de los semáforos son los propios procesos activos los que manejan las políticas de acceso a variables compartidas.

#### 2.4.3.2.2. Condición de sincronización en monitores

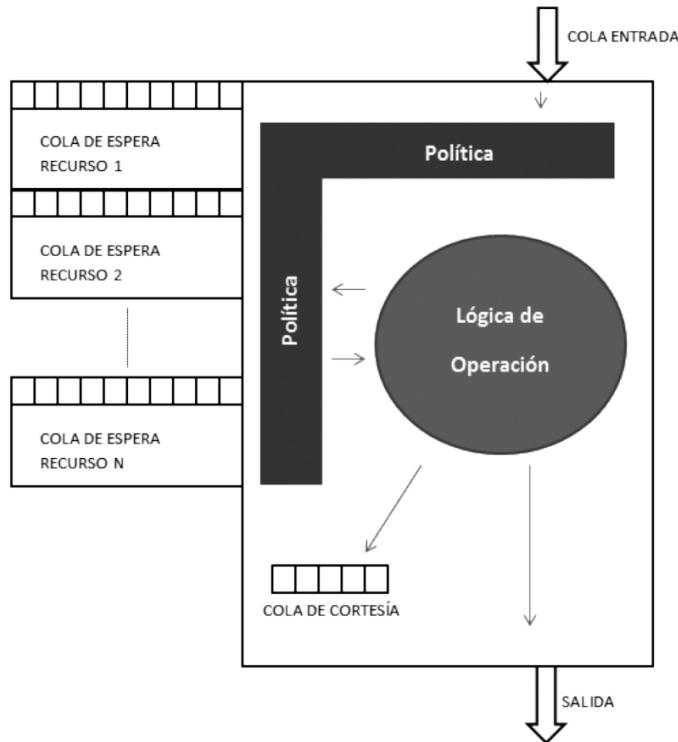
El procedimiento anterior sólo controla la exclusión mutua, es decir, pueden haber casos donde un proceso activo tenga acceso al monitor (ha obtenido la exclusión mutua al mismo) pero no puede seguir su ejecución debido a alguna razón, tal como un *buffer* lleno que no puede ser escrito. En estos casos, es necesario bloquear ese proceso y permitir que otro ingrese al monitor. Para realizar esto surgen nuevos componentes que deben formar parte del monitor:

- Variables de condición
  - Las mismas son declaradas en el monitor.
  - Deben ser privadas.
  - Tienen una cola *FIFO* asociada.
- Operaciones sobre las variables de condición
  - *Delay*
  - *Resume*
  - *Empty*

La operación *delay* se realiza sobre una variable de condición. Si se supone la existencia de una variable C, al realizar *delay(C)*, el proceso que la ejecutó libera el mutex del monitor, se bloquea y se envía al final de la cola asociada a la condición C. A diferencia de la operación *wait* que se utiliza en los semáforos, *delay* bloquea al proceso incondicionalmente.

La operación *resume*, cuando se realiza sobre una variable C, libera al primer proceso que ejecutó *delay(C)*. Si la cola está vacía, *resume* es una operación nula. Por otra parte, la función *empty* simplemente devuelve un valor boolean true si una cola se encuentra vacía o false en caso contrario.

Con lo dicho hasta este punto, se podría decir que si un proceso que se está ejecutando dentro del monitor ejecuta una operación *resume(C)*, se desbloqueará un proceso de esa cola que continuará con su ejecución dentro del monitor también. Esto lleva a una situación con dos procesos dentro del monitor, lo que violaría la exclusión mutua. Para evitar esto, el proceso que ejecuta el *resume* cederá la exclusión mutua al recién desbloqueado. Y espera en una cola diferente llamada **cola de cortesía** hasta que el proceso recién desbloqueado por el *resume(C)* termine su ejecución teniendo preferencia por sobre los procesos esperando en otras colas.

FIGURA 2.10: Estructura interna de un monitor<sup>1</sup>.

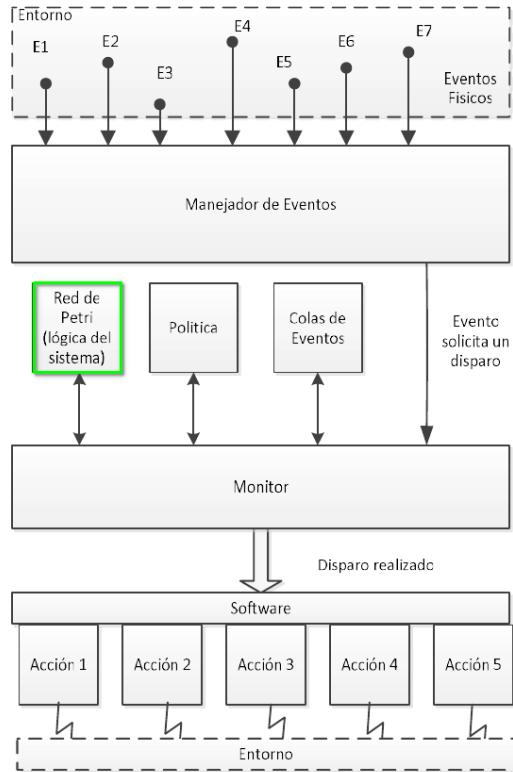
#### 2.4.3.3. Implementación de monitores con redes de Petri

Es posible ver a un monitor formado por dos secciones: primero, la referida a la política de colas que se debe ejecutar para lograr que sólo un proceso esté en el monitor, que se bloqueen los procesos que no tienen los recursos y que se desbloqueen los que obtuvieron los recursos, y segundo, la lógica con que se administran los recursos.

En la Figura 2.10 se puede observar que existe una cola de entrada, para los procesos que aún no ingresaron al monitor y desean hacerlo, una serie de colas, una por cada recurso (cada condición de sincronización) y una cola de cortesía para que proceso dentro del monitor pueda, de manera segura, ceder la exclusión mutua al cambiar el estado de un recurso.

*Una red de Petri puede realizar el trabajo de la lógica del monitor*, es decir, la administración y sincronización de recursos disponibles; esto es cuando el vector de estado que resultó del disparo no tiene componentes negativas es porque los recursos están disponibles, el disparo de la transición solicitada conduce a un nuevo estado valido. De no ser así, en caso de existir algún valor negativo en el nuevo vector de estado, se llegó a un estado no válido que indica que el recurso no está disponible. Además el vector de estado indica si el disparo ha devuelto o tomado recursos. Si la cantidad de tokens para un recurso dado disminuye, significa que se han tomado recursos, en caso contrario, que se han devuelto recursos [17, 18].

<sup>1</sup>Figura adaptada del Proyecto Integrador *Desarrollo de IP cores con procesamiento de Redes de Petri Temporales para sistemas multicore en FPGA* [17, 18].

FIGURA 2.11: Monitor<sup>2</sup>.

Por lo tanto, el monitor integra la red de Petri (lógica), la política y las acciones, conformando un sistema heterogéneo. La importancia de la metodología aquí planteada radica en desacoplar la lógica de la política y las acciones, con el fin de obtener un sistema resultante modular, simple, mantenable y verificable.

En la Figura 2.11 se expone la arquitectura modular de un sistema reactivo y guiado por eventos. Para el interés de este proyecto sólo se investigó sobre la lógica del sistema y específicamente sobre como desbloquear las redes de Petri (del tipo S<sup>3</sup>PR) que la representa.

## 2.5. S<sup>3</sup>PR

Se define la clase de los procesos secuenciales simples (S<sup>2</sup>P); luego, se extiende para modelar el uso de recursos (la clase de S<sup>2</sup>PR) y, finalmente, se define la clase de sistemas de procesos secuenciales simples con recursos (S<sup>3</sup>PR) por la composición neta de S<sup>2</sup>PR a través de un conjunto de lugares comunes [11].

### 2.5.1. Definición de S<sup>2</sup>P

Un sistema de proceso secuencial simple (S<sup>2</sup>P) es una red de petri  $N = (P \cup \{P^0\}, T, F)$  donde:

1.  $P \neq \emptyset$ ,  $p^0 \notin P$  ( $p^0$  llamada plaza idle);
2. N es una máquina de estado fuertemente conectada

<sup>2</sup>Figura adaptada del paper publicado por Micolini, Orlando & Ventre, Luis & Cebollada, Marcelo & Eschoyez, Maximiliano [17]

3. Cada circuito de N contiene la plaza  $p^0$ .

La tercera condición impone una propiedad de "terminación" a los procesos de trabajo que estamos considerando: si un proceso evoluciona, terminará.

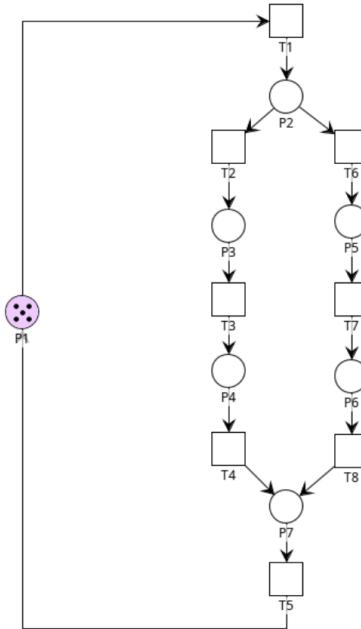


FIGURA 2.12: Red de Petri S<sup>2</sup>P<sup>3</sup>.

En la Figura 2.12 se puede observar la plaza idle ( $P_1$ ) destacada en lila.

Se define ahora un proceso secuencial simple con recursos (S<sup>2</sup>PR), como un S<sup>2</sup>P que necesita el uso de un recurso único en cada estado que no sea el estado idle. Debido a que las interacciones con el resto de los procesos en el sistema se realizarán compartiendo el conjunto de recursos, es natural suponer que en el estado idle no hay interacción con el resto del sistema y, por lo tanto, no se utiliza ningún recurso en este estado.

### 2.5.2. Definición de S<sup>2</sup>PR

Un sistema de proceso secuencial simple con recursos (S<sup>2</sup>PR) es una red de Petri  $N = \langle P \cup \{p^0\} \cup P_R, T, F \rangle$  tal que:

1. La subred generada por  $X = P \cup \{p^0\} \cup T$  es una S<sup>2</sup>P.
2.  $P_R \neq \emptyset$  y  $(P \cup \{p^0\}) \cap P_R = \emptyset$
3.  $\forall p \in P. \forall t \in \bullet p. \forall t' \in p \bullet . \bullet t \cap P_R = t' \bullet \cap P_R = \{r_p\}$
4. Las dos siguientes declaraciones son verificadas:
  - a)  $\forall r \in P_R. \bullet \bullet r \cap P = r \bullet \bullet \cap P \neq \emptyset$
  - b)  $\forall r \in P_R. \bullet r \cap r \bullet = \emptyset$
5.  $\bullet \bullet (p^0) \cap P_R = (p^0) \bullet \bullet \cap P_R = \emptyset$

<sup>3</sup>Figura adaptada del libro *Deadlock Resolution in Automated Manufacturing Systems* [11].

$P_R$ : conjunto de plazas de recursos.

$P$ : conjunto de plazas de estado.

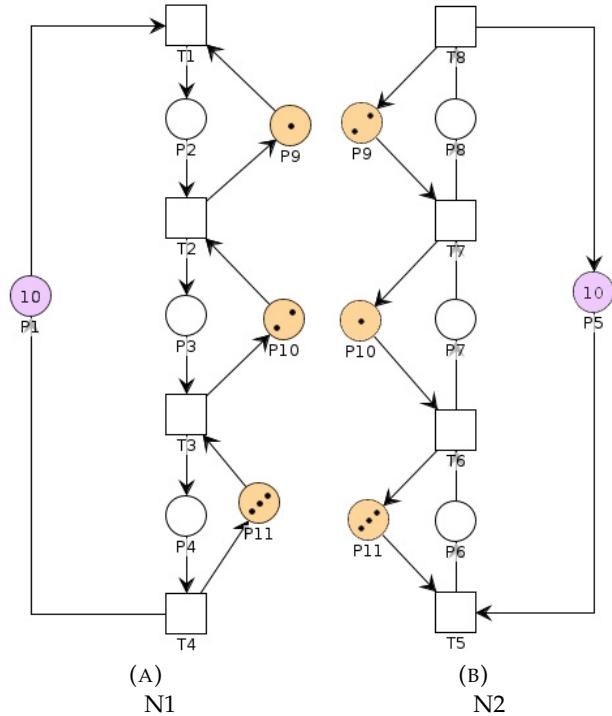


FIGURA 2.13: Redes de Petri S<sup>2</sup>PR. (A) RdP (N1,M1) - (B) RdP (N2,M2)<sup>4</sup>.

Esto se puede ver representado en las Figuras 2.13a y 2.13b. Las plazas idles ( $P_1, P_5$ ) destacadas en lila, mientras que las plazas recursos ( $P_9, P_{10}, P_{11}$ ) destacadas en naranja.

Para una plaza de estado dado  $p \in P$ , la plaza  $r \in P_R$  dado por la condición 3 en la definición modela el recurso utilizado en este estado. Para un  $r \in P_R$ , denotaremos  $H(r) = (\bullet \bullet r) \cap P$  conjunto de plazas complemento de  $r$  (estados que usan  $r$ ). La condición 4 en la definición anterior impone que dos estados adyacentes de un proceso de trabajo (WP) (ambos diferentes del estado inactivo) no pueden usar el mismo recurso. Esto no es una restricción, ya que desde la perspectiva de la vivacidad, dos estados adyacentes que usan el mismo recurso puede colapsar en un estado único, preservando las propiedades de comportamiento de la red.

Nótese que  $\bullet r$  representa las transiciones entrantes a las plazas  $r$   $\bullet \bullet r = \bigcup_{t \in \bullet r} \bullet t$  es el conjunto de plazas de entrada de todas las transiciones de entrada de la plaza  $r$ . De manera similar,  $r \bullet \bullet = \bigcup_{t \in r} \bullet t$  representa el conjunto de todas las plazas de salida de todas las transiciones de salida de la plaza  $r$ . Por ejemplo, en la Figura 2.14:  $\bullet p_9 = \{t_2, t_8\}$  y  $\bullet \bullet p_9 = \bullet t_2 \cup \bullet t_8 = \{p_2, p_{10}, p_8\}$ .  $p_9 \bullet = \{t_1, t_7\}$  y  $p_9 \bullet \bullet = t_1 \bullet \cup t_7 \bullet = \{p_2, p_{10}, p_8\}$ . Claramente  $\bullet \bullet p_9 = p_9 \bullet \bullet$

<sup>4</sup>Figura adaptada del libro *Deadlock Resolution in Automated Manufacturing Systems* [11].

Sea  $N = \langle P \cup \{p^0\} \cup P_R, T, F \rangle$  una  $S^2PR$ . Un marcado inicial  $m_0$  es llamado un marcado inicial aceptable para N si:

1.  $m_0(p^0) \geq 1$
2.  $m_0(p) = 0, \forall p \in P$
3.  $m_0(r) \geq 1, \forall r \in P_R$

Observe que una marca aceptable asigna al menos un token en la plaza idle (entonces, suponemos que, inicialmente, cada marca -token- de cada proceso está inactiva) y al menos un token en cada recurso, es decir, hay al menos una marca de cada recurso en el sistema. Está claro que si existe un recurso para el que no hay marca, el sistema no está bien definido, porque puede tener alguna secuencia de producción que no se puede llevar a cabo.

### 2.5.3. Definición de $S^3PR$

Entonces se puede concluir que un sistema de proceso secuencial simple con recursos  $S^3PR = \{N = \bigcup_{n=1}^k N_i = (P \cup P^0 \cup P_R, T, F)\}$  se define como la unión de un conjunto de redes, de tipo  $S^2PR = \{N_i = (P_i \cup p_i^0 \cup P_{Ri}, T_i, F_i)\}$ . Como se puede observar en la Figura 2.14.

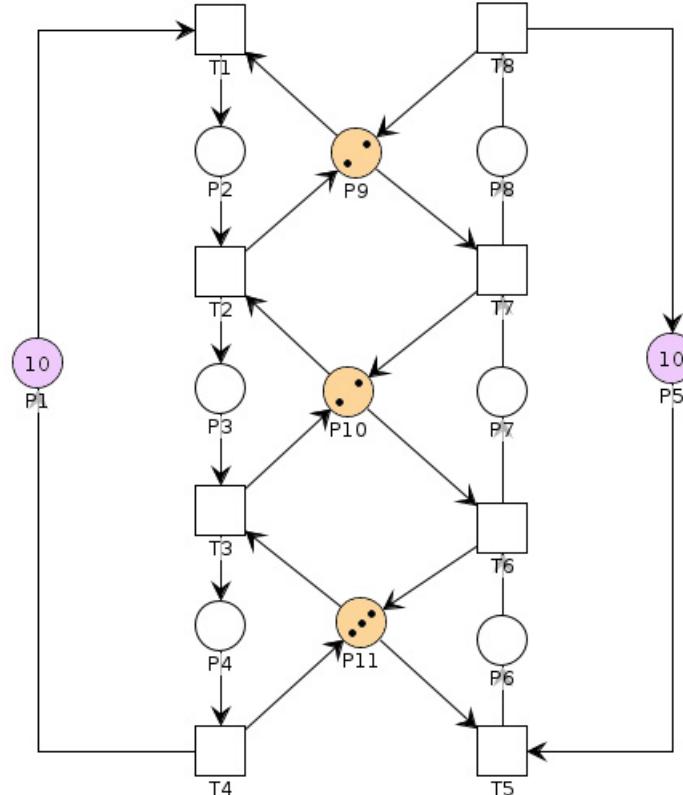


FIGURA 2.14: Composición de una red de Petri  $S^3PR$ .<sup>5</sup>

<sup>5</sup>Figura adaptada del libro *Deadlock Resolution in Automated Manufacturing Systems* [11].

Sean  $(N_1, M_1)$  y  $(N_2, M_2)$  dos redes de Petri con  $N_1 = (P_1, T_1, F_1, W_1)$  y  $N_2 = (P_2, T_2, F_2, W_2)$ , donde  $P_1 \cap P_2 = P_c \neq \emptyset$  y  $T_1 \cap T_2 = \emptyset$ .  $(N, M)$  con  $N = (P, T, F, W)$  es la red resultante de la unión entre  $(N_1, M_1)$  y  $(N_2, M_2)$  a través de compartir el conjunto de plazas  $P_c$  si (1)  $P = P_1 \cup P_2$ ,  $T = T_1 \cup T_2$ ,  $F = F_1 \cup F_2$ , y  $W(x, y) = W_i(x, y)$  si  $(x, y) \in F_i$ ,  $i = 1, 2$ ; y (2)  $\forall p \in P_1 \setminus P_c$ ,  $M(p) = M_1(p)$ ,  $\forall p \in P_2 \setminus P_c$ ,  $M(p) = M_2(p)$  y  $\forall p \in P_c$ ,  $M(p) = \max\{M_1(p), M_2(p)\}$ .

Por ejemplo: dos redes  $(N_1, M_1)$  y  $(N_2, M_2)$  se muestran en la Figura 2.13a y Figura 2.13b, respectivamente, donde  $P_1 = \{p_1 - p_4, p_9 - p_{11}\}$ ,  $T_1 = t_1 - t_4$ ,  $P_2 = p_5 - p_{11}$  y  $T_2 = t_2 - t_8$ . Donde  $P_1 \cap P_2 = \{p_9, p_{10}, p_{11}\}$  y  $T_1 \cap T_2 = \emptyset$ . En la Figura 2.14 se observa la red resultante de la composición de  $(N_1, M_1)$  y  $(N_2, M_2)$  es denotado como  $(N, M)$  donde  $P = P_1 \cup P_2 = \{p_1 - p_{11}\}$ ,  $T = T_1 \cup T_2 = \{t_1 - t_8\}$ ,  $M(p_1) = M_1(p_1) = 10$ ,  $M(p_2) = M_1(p_2) = 0$ ,  $M(p_3) = M_1(p_3) = 0$ ,  $M(p_4) = M_1(p_4) = 0$ ,  $M(p_5) = M_2(p_5) = 10$ ,  $M(p_6) = M_2(p_6) = 0$ ,  $M(p_7) = M_2(p_7) = 0$ ,  $M(p_8) = M_2(p_8) = 0$ ,  $M(p_9) = \max\{M_1(p_9), M_2(p_9)\} = 2$ ,  $M(p_{10}) = \max\{M_1(p_{10}), M_2(p_{10})\} = 2$ ,  $M(p_{11}) = \max\{M_1(p_{11}), M_2(p_{11})\} = 3$ .

## Capítulo 3

# Desarrollo

### 3.1. Desarrollo de la investigación

En la presente sección se busca explicar brevemente cómo se fue estructurando y desarrollando el algoritmo en cuestión.

Las redes implementadas en esta investigación fueron redes del tipo  $S^3PR$  dado que modelan la ejecución concurrente de procesos de trabajo. La finalización de alguno de estos puede iniciar más de una nueva operación. Como resultado de estas características dinámicas, pueden ocurrir dos situaciones: conflicto y deadlock.

El conflicto puede ocurrir cuando dos o más procesos requieren un recurso común al mismo tiempo. Por ejemplo, dos estaciones de trabajo pueden compartir un sistema de transporte común o necesitar acceso al mismo almacenamiento. Una forma sencilla de resolver el conflicto es asignar un nivel de prioridad a cada uno de los procesos.

El deadlock puede suceder al compartir dos recursos entre los dos procesos. En este caso, se puede alcanzar un estado en el que ninguno de los procesos puede continuar. Tenga en cuenta que uno de los procesos puede continuar si el conflicto se puede resolver, mientras que en el caso de deadlock no se puede hacer nada para que el sistema vuelva a funcionar.

### 3.2. Modelo de desarrollo

Para la elaboración del presente proyecto se optó por utilizar un modelo iterativo e incremental. A grandes rasgos, este tipo de modelo de desarrollo no es más que un conjunto de tareas agrupadas en pequeñas etapas repetitivas, las cuales inician con un análisis y finalizan con una versión nueva del algoritmo y sus conclusiones.

Se planifica un proyecto en distintos bloques temporales denominados iteraciones. Dentro de una iteración se repite un determinado proceso de trabajo sobre uno o varios objetivos, obteniéndose al final de la misma un resultado con más funcionalidades implementadas que el de la iteración anterior. La ventaja principal de este modelo es que no se debe esperar a que el sistema esté completo para que el mismo sea utilizable y operacional.

Para lograr esto, al realizar el análisis de una iteración, se especifican los objetivos que se esperan conseguir al finalizar la misma. Estos se establecen en función de los requerimientos, de los riesgos y de la evaluación de los resultados de las iteraciones precedentes. Se busca que en cada iteración los componentes logren evolucionar el

producto dependiendo de aquellos completados en las iteraciones antecesoras.

La implementación del algoritmo se realizó en tres iteraciones:

- **Iteración 1:** Procesamiento de los archivos '.html' y detección de estados de deadlock.
  1. Realizar la conversión de los archivos de formato '.html' exportados del Petrinator y su posterior procesamiento para que sean utilizados como entradas en el algoritmo.
  2. Detección de estados de deadlock basándose en los fundamentos teóricos (Capítulo 2).
  3. Una vez detectado, se evita alcanzar el mismo inhibiendo el disparo de la transición que lo desencadena.
- **Iteración 2:** Sifones en estado de deadlock y plaza de control.
  1. Detectar los sifones vacíos en estado de deadlock.
  2. Obtener el marcado inicial de cada sifón.
  3. Detectar la transición cuyo disparo lleva al camino del deadlock.
  4. Incorporar plaza y arco de control (salida) que evite el vaciado del sifón en cuestión.
- **Iteración 3:** Sifones mínimos en estado de deadlock y supervisor.
  1. Obtener los sifones mínimos vacíos (bad siphons) en estado de deadlock.
  2. Determinar el marcado y arcos (entrada y salida) del supervisor.
  3. Incorporar los supervisores que eviten el vaciado de los sifones en cuestión.

Las mismas serán tratadas con mayor profundidad a lo largo de este capítulo.

### 3.3. Iteración 1: Algoritmo v1.0

#### 3.3.1. Introducción

En esta iteración se hizo foco en analizar el grafo de alcanzabilidad de las redes, detectando los estados de deadlock alcanzables y trazando las secuencias de disparo que desencadenan a los mismos.

#### 3.3.2. Objetivos

- Conversión de archivos de formato '.html' (salida del software Petrinator) a archivos tipo '.txt' para facilitar la lectura de los datos en el algoritmo.
- Detectar el/los estado/s con deadlock y evitar el camino que desencadenan a ellos.

### 3.3.3. Desarrollo

#### 3.3.3.1. Herramienta a utilizar

El software seleccionado para obtener la información necesaria para el desarrollo del algoritmo fue Petrinator [4] debido a que el mismo fue desarrollado dentro del Laboratorio de Arquitecturas de Computadoras (LAC) lo que permitió solicitar modificaciones según las necesidades.

Sus principales características son:

- Una interfaz intuitiva que permite la creación, edición, almacenamiento y exportación de redes de Petri.
- Código abierto y bajo licencia GPL.

##### 3.3.3.1.1. Datos necesarios

Se utilizaron las funcionalidades del Petrinator para obtener y exportar la siguiente información de la red:

- Análisis de Invariantes (Invariant analysis)
  - T-invariantes
  - P-invariantes
- Matrices
  - Pre(+)
  - Post(-)
- Sifones y Trampas (Siphons and Traps)
  - Sifones (plazas que lo componen)
  - Trampas (plazas que lo componen)
- Grafo de Alcanzabilidad (Reachability graph)
  - Estados en deadlock
  - Marcado inicial de los sifones

#### 3.3.3.2. Conversión de datos

Como puente entre el software Petrinator y el algoritmo principal, se desarrolló un algoritmo de conversión en Python “html\_to\_txt”, el cual dados los archivos ‘.html’ exportados por el software trabaja sobre estos, con el fin de obtener los datos característicos de la red (mencionados anteriormente) en el formato adecuado de entrada que alimentará al algoritmo en desarrollo, en este caso matrices y vectores.

### 3.3.3.3. Construcción del algoritmo

Una vez realizada la carga y el filtrado de los datos exportados del software Petrinator, se llevó a cabo la detección de todos los estados de deadlock presentes en la red. Luego, se selecciona uno de estos y se trazan las secuencias de disparo que desencadenaron al mismo, se verifican todos los arcos de los estados que desencadenaron al mismo; si alguno de estos estados también estaban en deadlock se inhiben los mismos en el vector de sensibilización, esto se realiza recursivamente hasta encontrar algún estado que no presente deadlock dejando **solamente** este brazo habilitado en el vector de sensibilización, de esta manera nos aseguramos de no entrar en un camino que solo conduce al estado de deadlock.

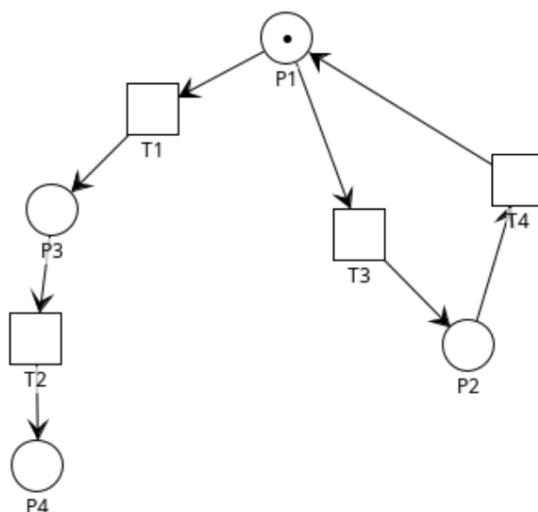


FIGURA 3.1: RdP con un estado de deadlock.

La red que se observa en la Figura 3.1 presenta 4 estados alcanzables y uno de ellos es un estado de deadlock.

- $S_0[1, 0, 0, 0]$
- $S_1[0, 0, 1, 0]$
- $S_2[0, 0, 0, 1]$
- $S_3[0, 1, 0, 0]$

Como se mencionó con anterioridad, se parte desde el estado  $S_0$  verificando que estado nos lleva al mismo. El único estado que lleva a este es el  $S_1$  (al ejecutarse la transición  $T_2$ ) por este motivo se inhibe esta transición del vector de sensibilizadas. Al realizar esto ahora el estado  $S_1$  se convierte en un estado de deadlock, y se realiza el mismo procedimiento mencionado con anterioridad inhibiendo la transición  $T_1$  que es la única que nos lleva a este estado a partir del estado  $S_0$ . De esta manera se obtiene una red sin deadlock dado que nunca se va tomar el camino de la izquierda, ya que se produjo la inanición de esta parte de la red.

Posteriormente se decidió aplicar la misma hipótesis sobre una red más compleja, con un mayor número de estados en su grafo de alcanzabilidad. Para esto se eligió la red representada en la Figura B.1

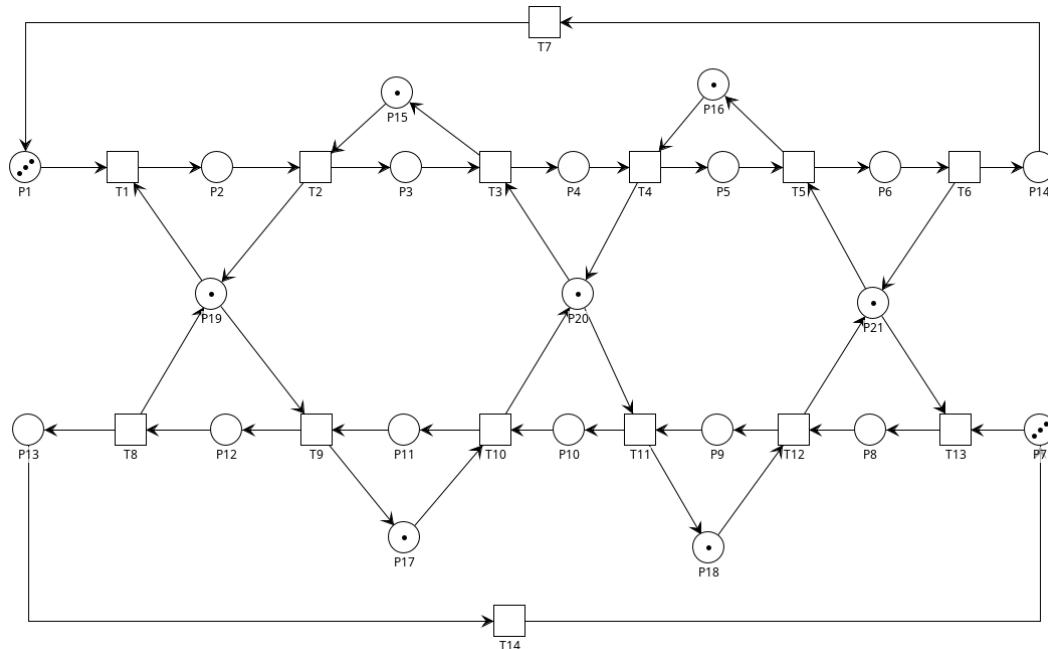
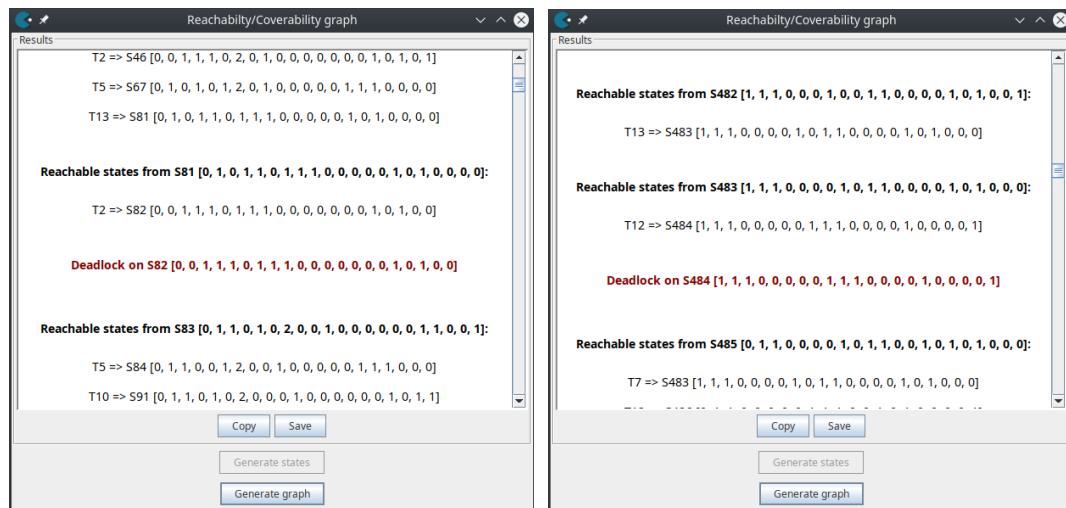
FIGURA 3.2: RdP Canal de Panamá.<sup>1</sup>

FIGURA 3.3: Estados de deadlock.

Como se puede observar en las Figuras 3.3a y 3.3b donde las transiciones  $T_2$  y  $T_{12}$  al ser ejecutadas llevan a los estados de deadlock; al deshabilitar la ejecución de las mismas la red sólo permite dos disparos (de las transiciones  $T_1$  y  $T_{13}$ ), luego de estos la red no se vuelve a ejecutar. Esta solución no es viable dado que la solución alcanzada al inhibir estas transiciones no representa la ejecución del modelo original.

<sup>1</sup>Figura adaptada del libro *An Algorithm for Deadlock Prevention Based on Iterative Siphon Control of Petri Net* [10].

### 3.3.4. Conclusiones

Si bien se alcanzó el objetivo planteado, este primer algoritmo no era escalable dado que si bien lograba resolver el deadlock para redes simples, en el caso de redes más complejas no llegaba a converger a una solución. Sin embargo, gran parte de las funcionalidades implementadas fueron utilizadas en las posteriores versiones del mismo.

## 3.4. Iteración 2: Algoritmo v2.0

### 3.4.1. Introducción

Al observar que la red presentaba un camino que llevaba al deadlock, se comenzó a analizar el porqué de este deadlock y se observó que el mismo se producía por el vaciado de al menos un sifón de la red. En esta nueva versión se hizo hincapié en evitar esta problemática.

### 3.4.2. Objetivos

- Encontrar los sifones vacíos en estado de deadlock.
- Evitar estados de deadlock.

### 3.4.3. Desarollo

En cuanto a los datos necesarios para el funcionamiento del algoritmo se reutiliza el código que permite la conversión de los mismos implementado en la versión 1.0 del algoritmo. A diferencia de la versión anterior, en vez de inhibir el arco imposibilitando la ejecución del mismo lo que se hace es lo siguiente:

1. Detectar el sifón vacío en el estado de deadlock.
2. Obtener el marcado del mismo en el estado inicial.
3. Detectar la transición cuyo disparo lleva a ese camino de deadlock.
4. Colocar una plaza “de control” cuyo marcado es la marca del sifón menos uno, evitando de esta manera que el sifón se vacíe (dejando su marcado al menos con un token).

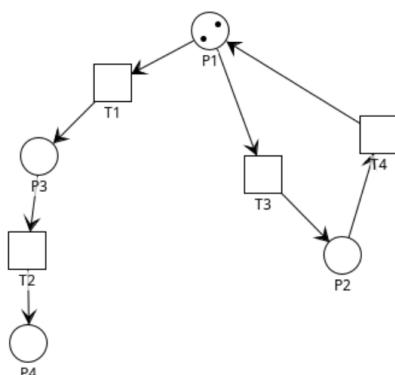


FIGURA 3.4: RdP con un estado de deadlock.

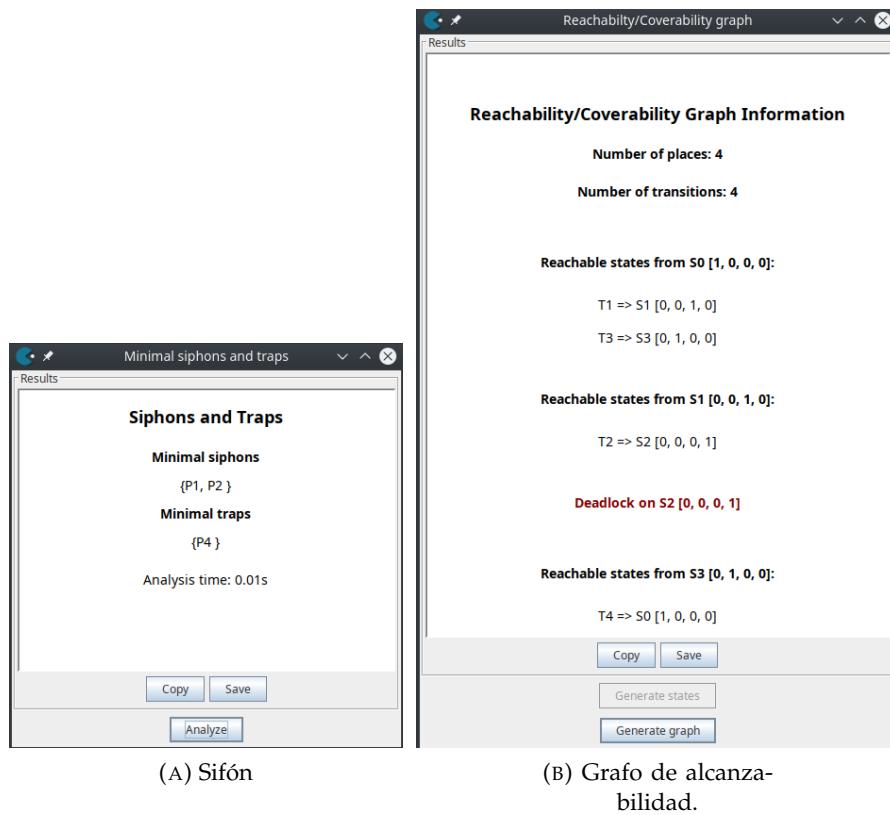


FIGURA 3.5: Análisis de la Figura 3.4.

En la Figura 3.4 se puede observar que si la red ejecuta a la transición  $T_1$  y luego  $T_2$  dos veces, la red se bloquea. Como se puede observar en la Figura 3.5b al ejecutar  $T_2$  desde el estado  $S_1$  conduce al deadlock. Al analizarla mediante el algoritmo y conociendo que el sifón que se presenta en la red está compuesto por  $\{P_1, P_2\}$  como se ve en la Figura 3.5a con un marcado inicial de 2, se obtiene un controlador que permite la ejecución de  $T_1$  una única vez, es decir, una vez ejecutada ésta la parte izquierda de la red no se podrá volver a ejecutar.

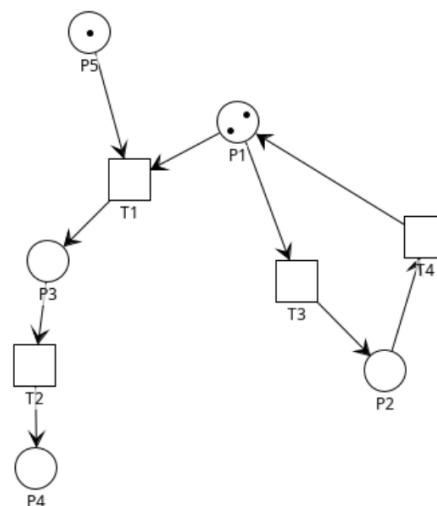


FIGURA 3.6: Control sobre la RdP.

Posteriormente se decidió aplicar la misma hipótesis sobre una red más compleja, con un mayor número de estados en su grafo de alcanzabilidad. Para esto se eligió la red representada en la Figura B.1

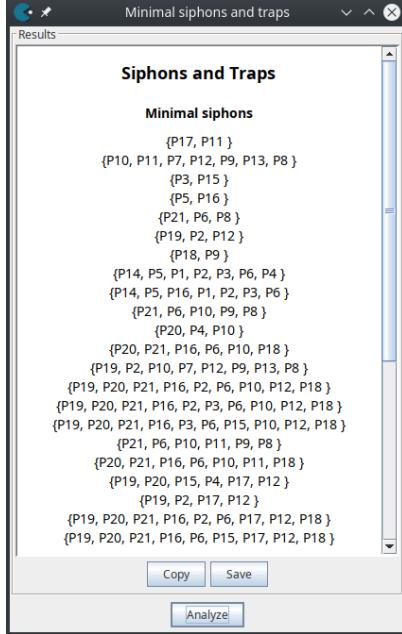


FIGURA 3.7: Sifones RdP Panamá.

Como se puede observar en la Figura 3.7 estos son los sifones presentes en la red. Al obtener los estados de deadlock 3.3a y 3.3b se indagó para verificar cuáles eran los sifones vacíos en los mismos. Para el estado número 82 el sifón que se vacía es el compuesto por las plazas  $\{P_4, P_{12}, P_{15}, P_{17}, P_{19}, P_{20}\}$  (3.8), mientras que para el estado número 484 el sifón vacío es el compuesto por las plazas  $\{P_6, P_{10}, P_{16}, P_{18}, P_{20}, P_{21}\}$  (3.9).

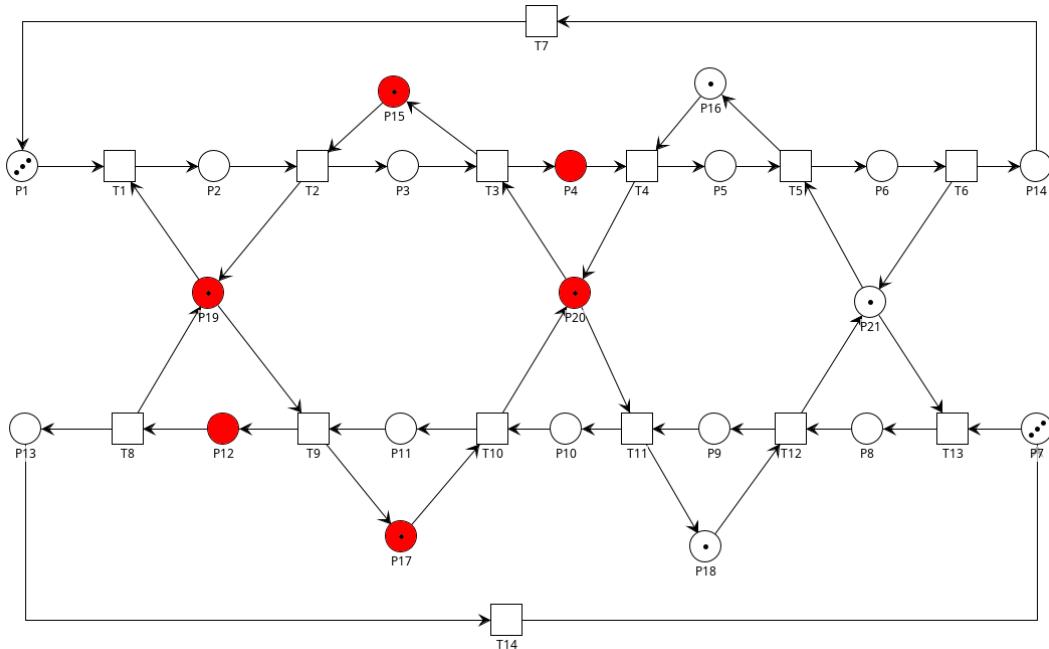


FIGURA 3.8: Sifón izquierdo.

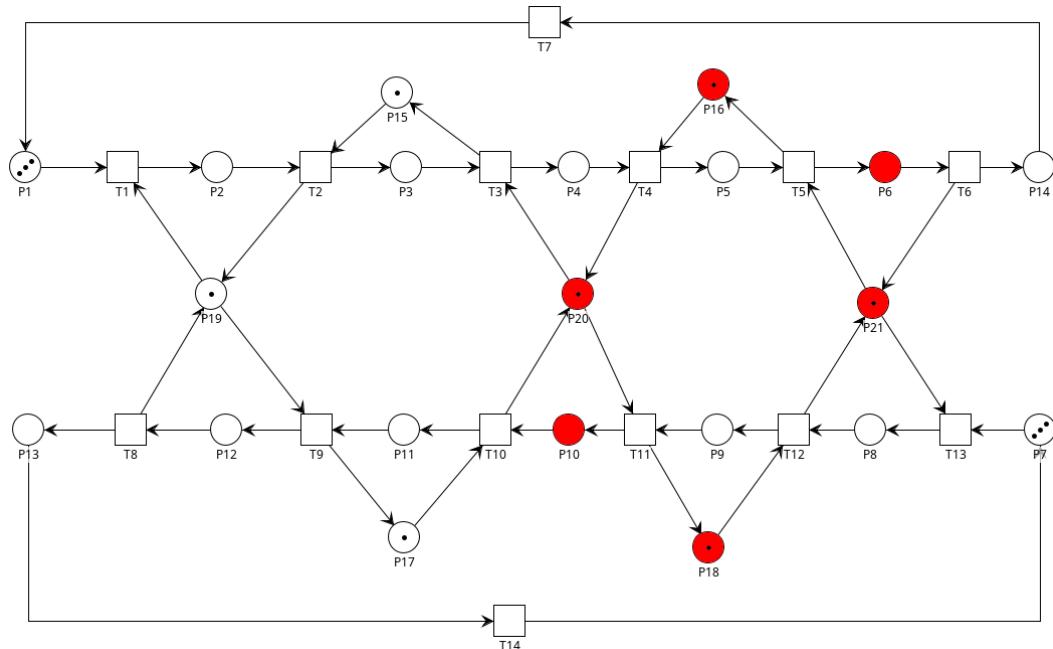
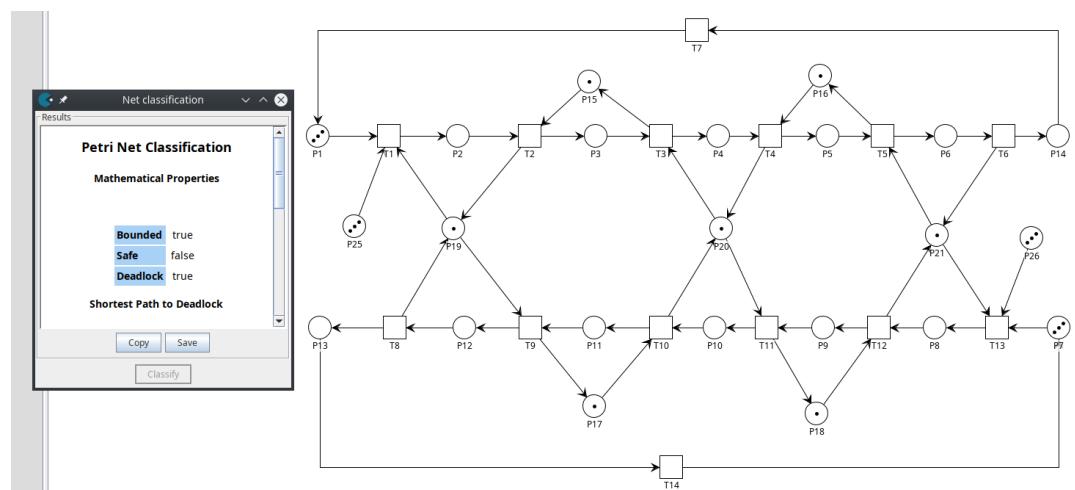


FIGURA 3.9: Sifón derecho.

En las Figuras 3.8 y 3.9 se pueden observar, destacados en color rojo, los sifones que se vacían en el estado de deadlock. De los mismos se obtiene el marcado inicial para luego colocar una plaza de control correspondiente a cada uno.

FIGURA 3.10: Control RdP Panamá, deadlock *true*.

### 3.4.4. Conclusiones

Nuevamente se alcanzó el objetivo propuesto, pero a diferencia de la versión anterior, cuando la red presenta un camino de deadlock este se ejecuta un número limitado de veces.

Y en coincidencia con la versión previa, sucede que la red presenta inanición y problemas de convergencia de la solución para redes grandes, como puede observarse en la Figura 3.10.

## 3.5. Iteración 3: Algoritmo v3.0

### 3.5.1. Introducción

Luego de una profunda investigación y adentrándose en el trabajo realizado por Ezpeleta et al. [8], el cuál expone una metodología en donde su principal idea es caracterizar situaciones de deadlock en términos de una marca igual a cero para ciertos sifones presentes en la red, pero no cualquiera de los sifones, sino los **sifones mínimos** mencionados en la Sección 2.3.4.

Para evitar que el sistema presente deadlock se propuso una política para la asignación de recursos basada en la adición de nuevas plazas de control a la red que impiden la presencia de sifones mínimos sin marcar. Estas nuevas plazas denominadas **supervisores**, están definidas por un marcado inicial y tres tipos de arcos; para obtener los primeros dos se tuvo en cuenta las fuentes investigadas, basándonos en la idea de un estado *idle* y de un conjunto complemento del sifón en cuestión. Mientras que para lograr el tercer arco se descubrió una relación entre el sifón a atacar y los T-invariantes encontrados en la red.

La adición de las plazas de control aseguran que el marcado de los sifones mínimos de la red sea al menos mayor o igual a uno para cada estado alcanzable, que es la condición necesaria para la prevención del deadlock. Esto se expresa matemáticamente de la siguiente manera:

$$\sum m(p_i) \geq 1, \text{ donde } p_i \in S \quad (3.1)$$

donde, la sumatoria de las marcas de todas las plazas que componen al sifón debe ser mayor o igual a uno; siendo  $p_i$  las plazas que pertenecen al sifón  $S$ .

### 3.5.2. Objetivos

Al igual que en las iteraciones anteriores el objetivo es evitar alcanzar estados de deadlock.

### 3.5.3. Desarrollo

Se parte de la hipótesis de obtener supervisores que impidan el vaciado de los **bad siphons mínimos**<sup>2</sup>, estos son aquellos sifones mínimos que desencadenan el estado deadlock al vaciarse, limitando el comportamiento de la red de Petri evitando que se alcancen dichos estados; y de esta manera preservar la vivacidad de una red determinada.

Ezpeleta propone una política para la asignación de recursos basada en la adición de supervisor(es). Esta política de control limita el comportamiento del sistema a un conjunto de estados de manera tal que, independientemente del estado que alcance el mismo, estamos seguros de que este no será una situación indeseable de deadlock. Tomando como base esto, se detallan en los siguientes puntos el progreso del algoritmo.

---

<sup>2</sup>Por simplicidad a partir de esta sección y a lo largo de todo el trabajo, la mención de *bad siphon* hará referencia a los *bad siphon mínimos* definidos.

### 3.5.3.1. Definición del supervisor

Para evitar alcanzar estados de deadlock, el sistema debe ser controlado por una nueva plaza denominada supervisor ( $V_s$ ), que está conectado con el proceso ( $G$ ) en un bucle cerrado (Figura 3.11). El mismo genera una secuencia de eventos discretos ( $s$ ) y lo envía al supervisor, el cual desencadena un conjunto de eventos permitidos ( $\gamma$ ) que suceden en el proceso  $G$  en el siguiente disparo. El conjunto  $\gamma$  depende de la secuencia  $s$  y no consta de los eventos que pueden llevar al proceso a un estado de deadlock en el siguiente paso.

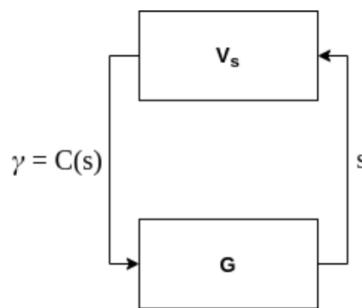


FIGURA 3.11: Retroalimentación entre el proceso y el supervisor<sup>3</sup>.

Para alcanzar el control antes mencionado se inicia analizando el grafo de alcancabilidad en busca de estados de deadlock. En caso de encontrarlos, se continúa con la búsqueda de bad siphons en cada uno de estos.

Ya con todos los estados en deadlock y sus sifones vacíos, se realiza un filtrado de los mismos descartando aquellos que se encuentren vacíos desde el marcado inicial de la red; estos no son de interés para nuestro análisis dado que una vez vacíos (por definición) estos permanecerán en ese estado sin modificar su comportamiento.

Una vez realizado el filtrado, se seleccionó uno de los sifones perteneciente a uno de los estados deadlock, y es a este, al que se buscó controlar.

El supervisor que va a controlarlo está compuesto por un conjunto de arcos y por una plaza con un marcado proporcionado por el marcado en el estado inicial (*idle*) del sifón menos uno. Esto se expresa matemáticamente de la siguiente manera:

$$m_0(V_s) = m_0(S_i) - 1 \quad (3.2)$$

El conjunto de arcos vincula la plaza mencionada con tres tipos de transiciones:

1. **Transiciones sensibilizadas en estado idle:** la ejecución de estas transiciones son las que extraen tokens del supervisor dado que el disparo de las mismas inicia los diferentes procesos que componen a la red, pudiendo tomar un camino que conduce al sifón (por lo que el token consumido será devuelto luego de una secuencia de disparos) ó tomar otro camino que no contemple al sifón y en tal caso las transiciones del ítem 3 serán las encargadas de devolver el token consumido al supervisor.
2. Para el segundo conjunto de transiciones es necesario definir un nuevo conjunto de plazas denominadas **complemento del sifón**, estas son aquellas plazas

<sup>3</sup>Figura adaptada del paper publicado por D. Kezić et al. [9].

que no forman parte del sifón pero para evolucionar en su marcado requieren del disparo de transiciones que se habilitan mediante el marcado de las plazas recurso que componen al sifón, es decir, hacen uso de estas. Es por esto que las transiciones de salidas al conjunto complemento del sifón son aquellas que le agregan tokens al supervisor.

3. Para definir este último conjunto de transiciones se realizó una investigación paralela dado que Ezpeleta no contemplaba este caso. En donde el último arco a tener en cuenta se obtuvo a partir de la relación entre el bad siphon a controlar y los T-invariantes presentes en la red, dependiendo del camino que tome la secuencia de disparos es necesario que este arco devuelva el token al supervisor.

Para esto es necesario verificar la presencia de un conflicto o bifurcación en la red entre T-invariantes, y en caso de existir, enfocarse en las transiciones involucradas en el mismo tal que al dispararse cualquiera de estas no alcancen al sifón, posterior a una secuencia de disparos, ya que el token que se le quitó al supervisor con el disparo de las transiciones idle, mencionadas en ítem 1, nunca volverá a este.

Para corregir esto, con el disparo de la transición conflictiva en cuestión, es necesario implementar el arco que devuelva el token consumido al supervisor. En caso de dispararse cualquiera de las transiciones del conflicto que posteriormente toman el camino hacia el sifón, no es necesario este arco ya que implica que la extracción por parte de las transiciones idle al supervisor va ser utilizada por la subred formada por el sifón a controlar y los arcos mencionados en el ítem 2, los cuales devolverán el token al supervisor.

Tomando la red definida por Ezpeleta se pueden observar los tres tipos de transiciones que conforman el supervisor. El sifón a controlar, en este caso, es el compuesto por las plazas  $\{P_7, P_8, P_9, P_{10}\}$ .

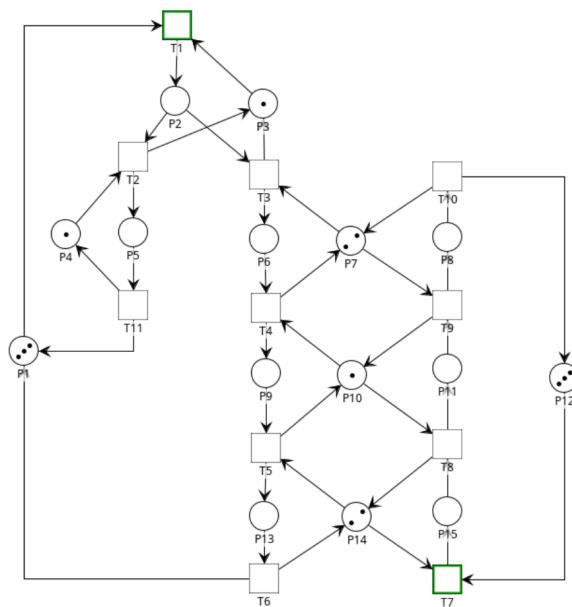


FIGURA 3.12: RdP Ezpeleta <sup>4</sup>.

<sup>4</sup>Figura adaptada del paper publicado por Ezpeleta et al. [8].

En la Figura 3.12 se observan las transiciones sensibilizadas en el estado inicial  $\{T_1, T_7\}$ , definiendo de esta manera los primeros arcos que componen al supervisor.

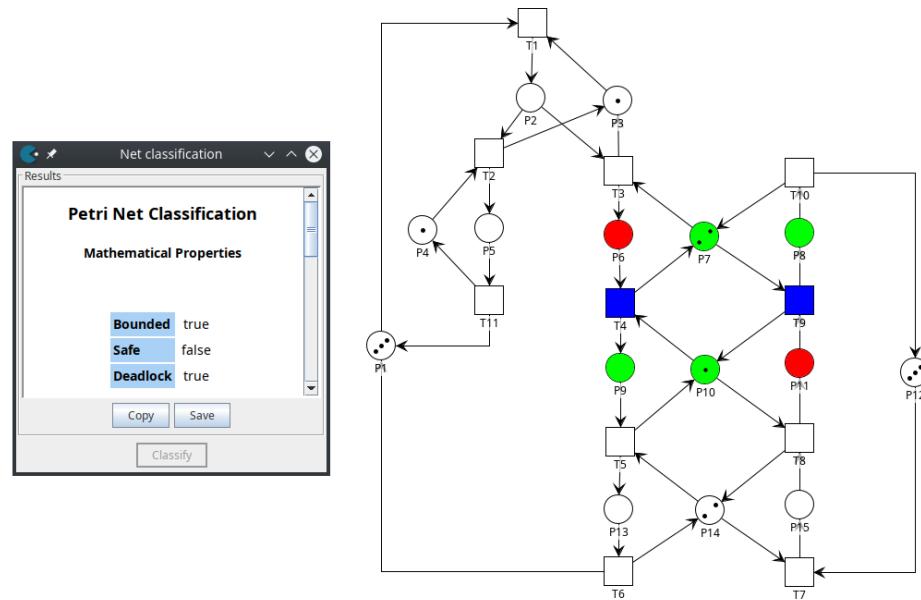


FIGURA 3.13: RdP Ezpeleta con sifón a controlar y sus plazas complemento.

Observando la Figura 3.13 se distingue en verde las plazas que conforman el sifón a controlar y en rojo las plazas complemento del mismo; mientras que en azul las transiciones  $\{T_4, T_9\}$  que conectarán con el supervisor mediante el segundo conjunto de arcos.

Para definir el tercer tipo de transiciones es necesario tener en cuenta los T-invariantes de la red dado que estos nos permiten observar los diferentes caminos que puede tomar la red en la evolución de los disparos de las transiciones de la misma.

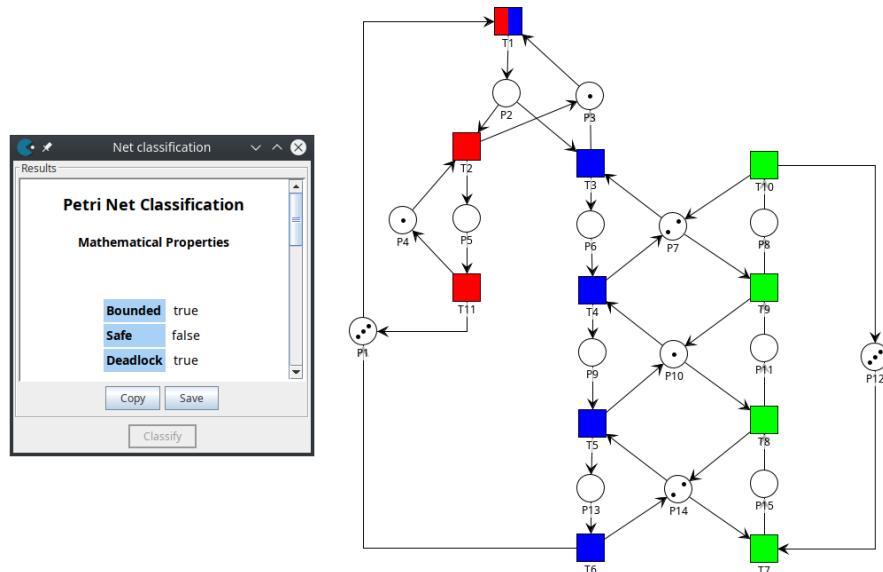


FIGURA 3.14: RdP Ezpeleta y sus T-invariantes.

Como ilustra la Figura 3.14, existen tres invariantes que componen la red (en distintos colores) y la plaza  $P_2$  involucrada en un conflicto, dado que su marcado habilita dos posibles caminos de T-invariantes, pudiendo tomar uno u otro. En caso de que se proceda con el disparo de la transición  $T_2$ , produciendo de esta manera la ejecución de las transiciones que componen al T-invariante rojo, el cual no afecta el marcado del sifón en cuestión y por lo que su ejecución devolverá el token al supervisor (se conectaría con el supervisor mediante el tercer conjunto de arcos).

Según la definición de la Sección 3.5.3.1, el supervisor queda conformado de la siguiente manera:

- El marcado =  $M(S_i) - 1 = \{M(P_7) + M(P_8) + M(P_9) + M(P_{10})\} - 1 = 2$
  - Transiciones output =  $\{T_1, T_7\}$
  - Transiciones input =  $\{T_2, T_4, T_9\}$

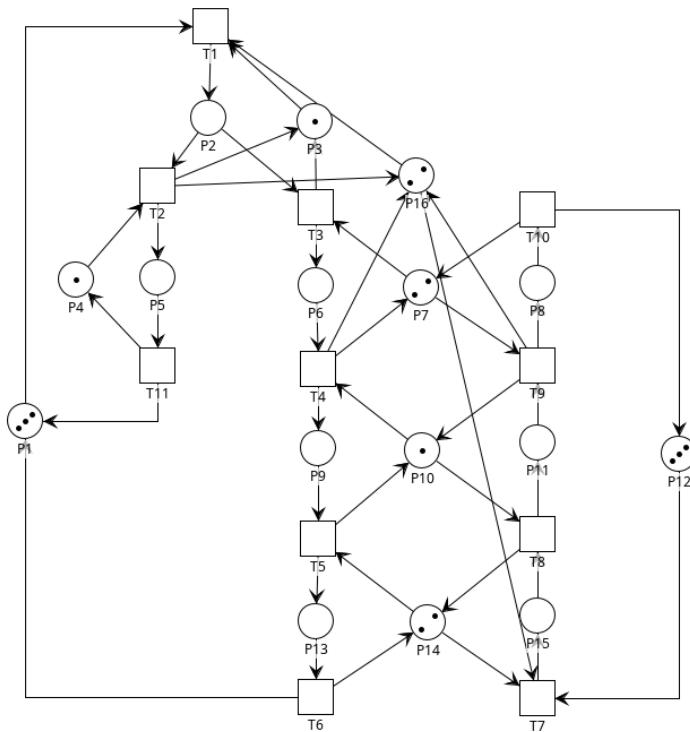


FIGURA 3.15: Colocación de un supervisor en RdP Ezpeleta.

Por último, en la Figura 3.15, se observa la red al controlar uno de los bad siphon, este control se logra al colocar el supervisor ( $P_{16}$ ) con su marcado y arcos respectivos.

El análisis completo de la red se llevará a cabo en la Sección posterior 3.5.6.2 dado que la red aún presenta deadlock.

### 3.5.4. Implementación del algoritmo

En esta sección se desarrollan los 5 pasos que modelan el algoritmo en la determinación del supervisor.

### 3.5.4.1. Desarrollo

1. Obtener los sifones vacíos en el estado inicial; estos sifones deben ignorarse dado que una vez vacíos permanecerán así por el resto de los estados alcanzables.
2. Obtener los estados en deadlock con sus respectivos sifones vacíos.
3. Seleccionando uno de los sifones mencionados en el ítem anterior:
  - a) Se obtiene su marcado inicial para posteriormente definir el marcado de su correspondiente supervisor.
  - b) Se localizan las transiciones sensibilizadas en el estado idle (para el marcado inicial).
  - c) Se obtienen las plazas complemento del mismo.
    - i. Se buscan las transiciones que quitan y agregan tokens a estas plazas.
    - ii. Las transiciones que agregan más tokens de los que quitan al sifón son las que nos interesan.
  - d) Se verifica si hay transiciones en conflicto, de ser así se utilizan los T-invariantes para verificar si la ejecución de la misma se encuentra en el camino de las plazas del sifón. De no ser así, estas transiciones serán también de interés.

Las transiciones destacadas en los ítems anteriores van a ser las que van a incorporar y extraer tokens del supervisor, como se mencionaron en la iteración 2.

4. Agregar una nueva plaza de control (perteneciente al supervisor) a la red puede producir un nuevo sifón mínimo no controlado y un nuevo estado de bloqueo. Por lo tanto, debemos volver al punto 1 calculando nuevamente el árbol de alcanzabilidad y repetir todo el algoritmo, atacando la totalidad de los bad siphon hasta alcanzar una red viva.

El algoritmo finaliza cuando no es posible encontrar un nuevo punto muerto en la red de Petri, es decir, se resuelve el deadlock de la misma.

Sin embargo, puede darse la situación en donde el algoritmo no converge a una solución dado que sugiere supervisores con marcado igual a 0 o supervisores ya colocados.

5. En caso, de que los pasos anteriores no convergen a una red libre de deadlock se realiza un análisis de división de la misma y se ataca cada subred resultante por separado, es decir, se debe ejecutar el algoritmo desde el paso 1 al 4 para cada subred, para luego reunir las soluciones en la red de Petri original.

La división se realiza teniendo en cuenta los T-invariantes y su relación con los bad siphon.

Para esto se deben tener en cuenta el siguiente factor:

- a) En caso de que la división de la red resulte en una de las subredes que contempla el conflicto en su totalidad (caso red POPN (Sección 3.5.6.3)), las soluciones de ambas subredes se pueden unir conservando la vivacidad de la red sin problema.

En caso de tener supervisores en común entre las subredes, es decir, tienen el mismo conjunto de arcos entrantes y salientes, al momento de definir el marcado del mismo en la red original, el supervisor debe tomar el valor del marcado del menor de ellos dado que no tiene que permitir el vaciado del sifón con menos marcas.

Las fórmulas para calcular el supervisor para un sifón son las siguientes:

1.  $m(V_s) = m(BS_i) - 1$
2.  $Arco_1 = \{(V_s, t) / t \in P_0\bullet\}$
3.  $Arco_2 = \{(t, V_s) / t \in C_s\bullet\}$
4.  $Arco_3 = \{(t, V_s) / t \in conflicto \wedge t \notin T_{inv_{BS_i}}\}$

siendo:

1.  $V_s$  = plaza supervisor
2.  $P_0$  = plazas marcadas idle
3.  $C_s$  = complemento sifón
4.  $BS_i$  = bad siphon
5.  $t$  = conjunto de transiciones

En la Figura 3.16 se puede observar, a modo de ejemplo de aplicación, la salida del algoritmo por consola para la ejecución de la red de Petri Panamá, analizada en profundidad en la Sección 3.5.6.1.

```
Path del archivo de Estados (html): pa_panama.html
Path del archivo de Matrices I(html): ma_panama.html
Path del archivo de Sifones(html): si_panama.html
Path del archivo de invariantes (html): in_panama.html
Sifones vacios en idle []
Estados con deadlock [82, 484]
Cantidad de estados con deadlock 2
Estado deadlock, sifon asociado al deadlock y su marcado [[82, 19, 4], [484, 2, 4]]
Cantidad de sifones vacios: 2

Sifon a controlar: 19
Marcado del supervisor 3
Transicion output: 1
Transicion output: 13
Transicion input: 5
Transicion input: 11

Sifon a controlar: 2
Marcado del supervisor 3
Transicion output: 1
Transicion output: 13
Transicion input: 3
Transicion input: 9
```

FIGURA 3.16: Ejecución del algoritmo v3.0.

- I. Las primeras 4 líneas son para la conversión de los datos exportados mediante el software Petrinator:

- *pa\_panama.html* contiene el grafo de alcanzabilidad.
  - *ma\_panama.html* contiene la matriz pre y post.
  - *si\_panama.html* contiene los sifones y las trampas.
  - *in\_panama.html* contiene los invariantes de plaza y de transición.
- II. En este caso no fue necesario descartar ningún sifón en el análisis dado que en el estado inicial todos los sifones se encuentran marcados.
- III. Se obtienen los estados de deadlock con sus respectivos sifones vacíos y el marcado de los mismos.
- IV. Como se puede observar el algoritmo brinda todos los posibles supervisores para el control de los sifones vacíos en estados de deadlock.

Se toma uno de los supervisores, se incorpora a la red en el software Petrinator realizando el análisis correspondiente en búsqueda de verificar que el deadlock de la red haya desaparecido; de no ser así se exportan nuevamente los archivos y se realiza la ejecución del algoritmo nuevamente. Y así iterativamente hasta lograr que el estado de deadlock de la red desaparezca.

#### 3.5.4.2. Pseudocódigo

Definiendo:

- $C_S$ : complemento del sifón
- $t$ : conjunto de transiciones
- $V_S$ : plaza supervisor
- Cantidad de sifones = cantidades de sifones total del sistema
- SD: state deadlock
- S: conjunto de sifones
- BS: conjunto de bad siphons

---

**Pseudocódigo 1** Búsqueda de bad siphon a controlar (v3)

---

**Input:** RdP ( $N, M_0$ ) de tipo S<sup>3</sup>PR.

**Output:** bad siphon.

- 1: Generar el grafo de alcanzabilidad  $G(N, M_0)$  de la RdP.
  - 2: Obtener matrices  $I^+$ ,  $I^-$ , invariantes, trampas y sifones.
  - 3: **for**  $i: 0$  **to** cantidad de estados **do**
  - 4:     **if** estado = estado en deadlock **then**
  - 5:          $SD_j \leftarrow estado_i$
  - 6:     **if** estado = idle **then**
  - 7:         **for**  $i: 0$  **to** cantidad de sifones **do**
  - 8:             **if**  $M(S_i) = 0$  **then**
  - 9:                  $BS_{idle} \leftarrow S_i$
  - 10: en  $SD[0]$
  - 11: **for**  $i: 0$  **to** cantidad de sifones **do**
  - 12:     **if**  $M(S_i) = 0$  **then**
  - 13:          $BS_{SD} \leftarrow S_i$
  - 14: Se eliminan de  $BS_{SD}$  aquellos que estén en  $BS_{idle}$
  - 15:  $BS_{SD}[0] \rightarrow$  control de bad siphon usando **Algoritmo 2**
- 

---

**Pseudocódigo 2** Búsqueda de supervisor que controle bad siphon (v3)

---

**Input:** RdP ( $N, M_0$ ) de tipo S<sup>3</sup>PR.

**Output:** supervisor.

- 1: Agregar plaza de control  $VS_i$  con  $M(VS_i) = M(BS_{SD}) - 1$
  - 2: **if** estado = idle **then**
  - 3:     Aregar arco  $(VS_i, t) \forall t \in p\bullet$
  - 4: Aregar arco  $(t, VS_i) \forall t \in C_S\bullet$
  - 5: Aregar arco  $(t, VS_i) \forall t \in conflicto \wedge t \notin T - invariante_{BS}$
- 

El algoritmo iterativo para lograr una red de Petri sin deadlock se muestra en la Figura 3.17

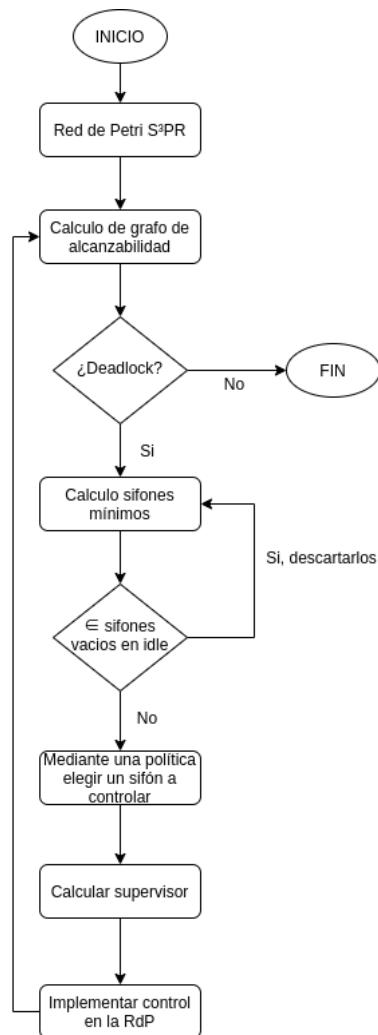


FIGURA 3.17: Ejecución del algoritmo v3.0.

### 3.5.5. Criterio de elección del supervisor

Al ejecutar el algoritmo, se obtiene una lista de supervisores a colocar dependientes al bad siphon que se va a controlar. El criterio de elección de qué supervisor agregar se realiza teniendo en cuenta:

- Afecte a un sifón mínimo.
- Cantidad de veces que aparece el supervisor en la lista.
- En caso de haber más de un supervisor sugerido para el control de un bad siphon, se elige aquel que presente la menor cantidad de *inputs*.

### 3.5.6. Ejecución en diferentes escenarios

En esta sección se busca explicar cómo fue progresando el algoritmo a medida que se implementó en diferentes casos de redes de Petri, dado que en cada nueva red se encontraban situaciones diferentes que debían tenerse en cuenta y cada una implicó una extensión más para el algoritmo final.

Estas extensiones se deben a que en cada una de las redes analizadas la relación que

presentaban sus T-invariantes con el bad siphon a controlar era diferente, esto fue lo que permitió generalizar el algoritmo de manera de contrarrestar el deadlock en cada una de las variantes.

### 3.5.6.1. Caso Panamá

Esta red modela un sistema de tráfico marino. El problema es la posibilidad de interbloqueo entre los barcos que circulan a través de un sistema de canales y dársenas (tres canales y cuatro dársenas). Los barcos circulan hacia la derecha o izquierda, esperando a cada lado del sistema de canales su derecho a paso. La capacidad de cada canal y dársena es de un barco a la vez, por lo que un barco no puede ingresar a un canal o dársena ocupado, estando obligado a esperar a que se desocupe.

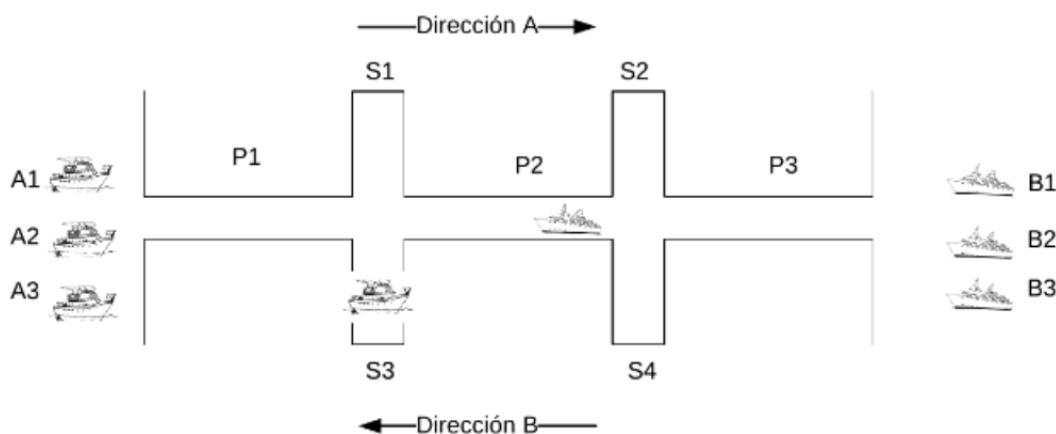


FIGURA 3.18: Modelo del Canal de Panamá<sup>5</sup>.

#### 3.5.6.1.1. Características generales

- Las cuatro dársenas están representadas por las plazas  $\{P_{15}, P_{16}, P_{17}, P_{18}\}$ .
- Los tres canales se denotan por las plazas  $\{P_{19}, P_{20}, P_{21}\}$ .
- Mientras que los barcos por las marcas de las plazas  $\{P_1, P_7\}$ .

<sup>5</sup>Figura adaptada del libro *An Algorithm for Deadlock Prevention Based on Iterative Siphon Control of Petri Net* [10].

### 3.5.6.1.2. Análisis estructural

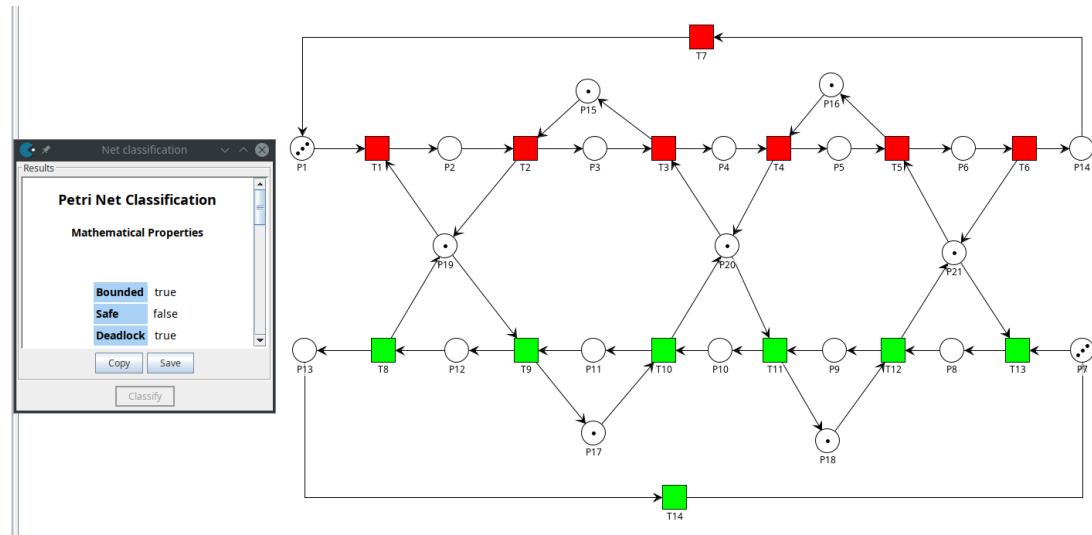


FIGURA 3.19: RdP Canal de Panamá<sup>6</sup> y sus T-invariantes.

En la Figura 3.19, la red no presenta un conflicto entre los T-invariantes, representados en rojo y verde. Se ejecuta el algoritmo desde el punto 1 al 4 permitiendo encontrar los supervisores sin necesidad de subdividir la red; incluso realizar esta acción en esta red no tendría mucho sentido dado que los T-invariantes por separado no representan el comportamiento de la red en su totalidad.

### 3.5.6.1.3. Red controlada

Una vez ejecutado el algoritmo, se obtienen los supervisores a agregar para controlar la red solucionando el deadlock, estos están representado por las plazas  $P_{22}$  y  $P_{23}$  en la Figura 3.19.

Supervisor	Marcado	Transiciones input	Transiciones output	Bad Siphon Controlado
$P_{22}$	3	$\{T_5, T_{11}\}$	$\{T_1, T_{13}\}$	$\{P_6, P_{10}, P_{16}, P_{18}, P_{20}, P_{21}\}$
$P_{23}$	3	$\{T_3, T_9\}$	$\{T_1, T_{13}\}$	$\{P_4, P_{12}, P_{15}, P_{17}, P_{19}, P_{20}\}$

CUADRO 3.1: Supervisores: RdP Panamá.

<sup>6</sup>Figura adaptada del libro *An Algorithm for Deadlock Prevention Based on Iterative Siphon Control of Petri Net* [10].

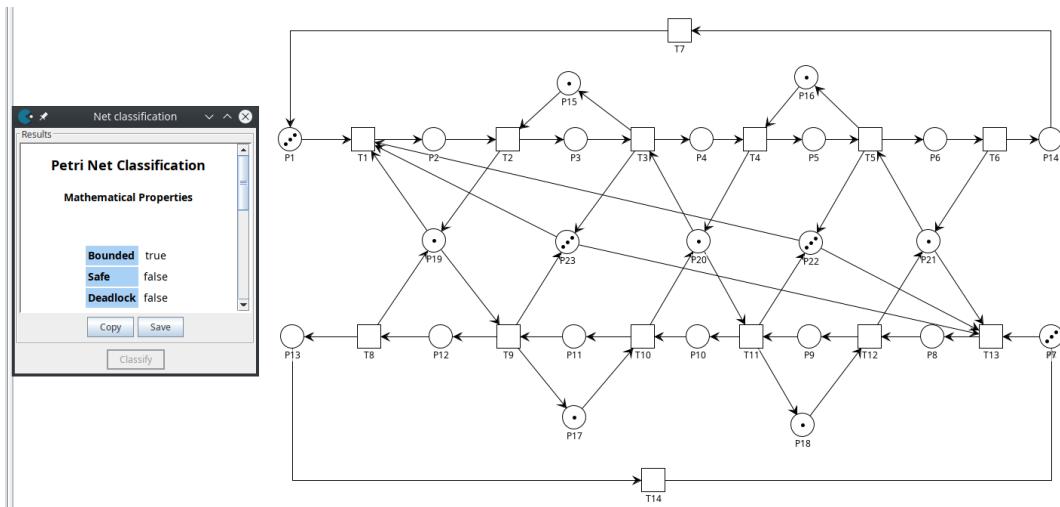


FIGURA 3.20: RdP Canal de Panamá controlada.

### 3.5.6.2. Caso Ezpeleta

Esta red modela la ejecución concurrente de procesos de trabajo en FMS, representando un sistema donde se ejecutan dos tipos de procesos de trabajo.

En la red existen plazas que simulan la disponibilidad de recursos (5) y un control incorrecto de estos en la ejecución de los procesos de trabajo, puede conducir a situaciones de deadlock.

### **3.5.6.2.1. Características generales**

- Presenta conflicto entre T-invariantes.
  - Los recursos de la red están representados por las plazas  $\{P_3, P_4, P_7, P_{10}, P_{14}\}$ .

### **3.5.6.2.2. Análisis estructural**

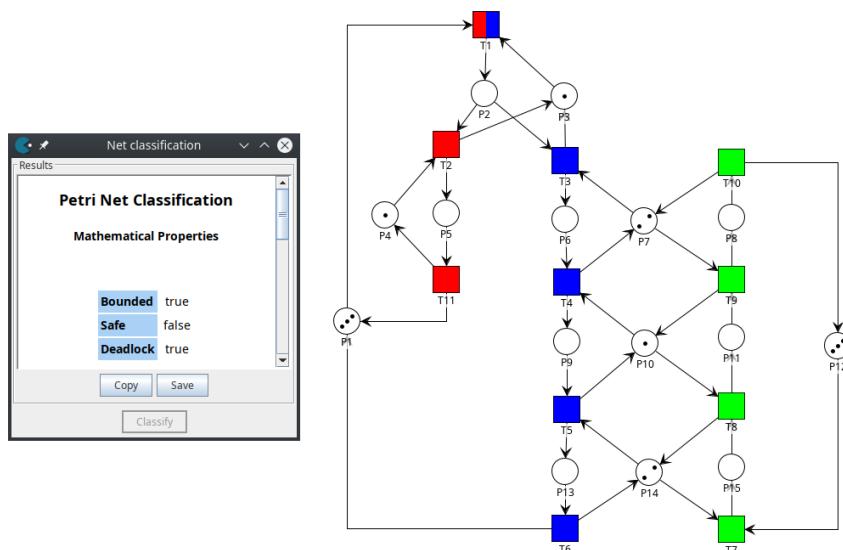


FIGURA 3.21: RdP Ezpeleta<sup>7</sup> y sus T-invariantes.

En la Figura 3.21, la plaza  $P_2$  forma parte de un conflicto permitiendo la ejecución de un subciclo de la red u otro, pudiendo seguir dos T-invariantes diferentes (rojo o azul en este caso).

Es por esto que fue necesario ejecutar el algoritmo de forma completa, es decir, desde el punto 1 al 5, contemplando la división de la red dado que los primeros 4 pasos no lograron alcanzar una red libre de deadlock.

En la división resultaron dos subredes, ambas preservando el conflicto (plaza y transiciones que lo conforman).

### Subred izquierda

Una de las subredes (Figura 3.22) no fue necesario controlarla dado que no presentaba deadlock.

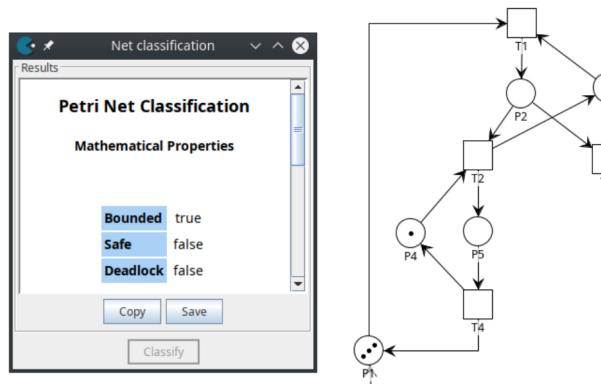


FIGURA 3.22: Subred izquierda Ezpeleta.

### Subred derecha

Mientras que la segunda subred (Figura 3.23) si fue necesario encontrar los supervisores que resuelvan el deadlock.

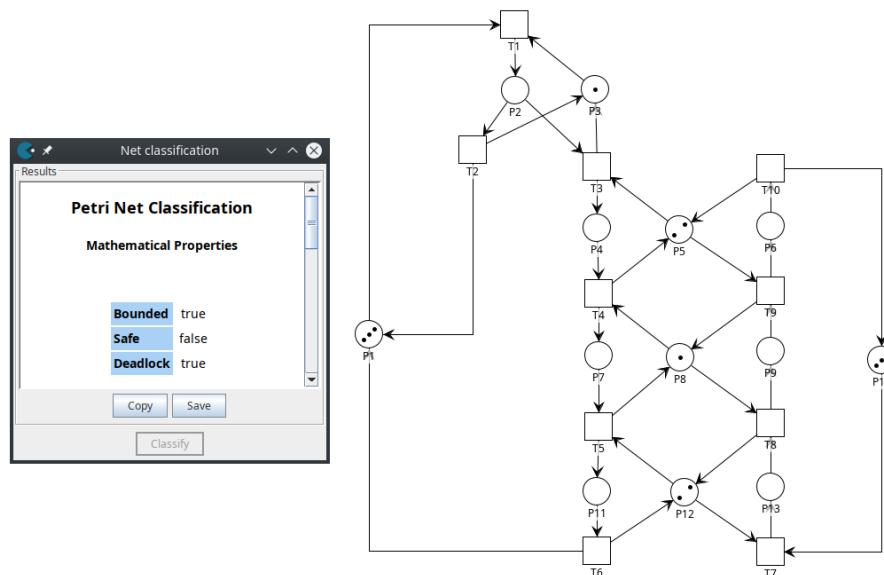


FIGURA 3.23: Subred derecha Ezpeleta.

<sup>7</sup>Figura adaptada del paper publicado por Ezpeleta et al. [8].

### Control subred derecha

Una vez ejecutados los 4 primeros pasos del algoritmo, sobre la subred en cuestión, se obtuvieron los supervisores a agregar para controlar la misma, resolviendo el problema de punto muerto.

Supervisor	Marcado	Transiciones input	Transiciones output	Bad Siphon Controlado
$P_{15}$	4	$\{T_2, T_5, T_9\}$	$\{T_1, T_7\}$	$\{P_5, P_6, P_8, P_{11}, P_{12}\}$
$P_{14}$	2	$\{T_2, T_5, T_8\}$	$\{T_1, T_7\}$	$\{P_8, P_9, P_{11}, P_{12}\}$
$P_{13}$	2	$\{T_2, T_4, T_9\}$	$\{T_1, T_7\}$	$\{P_5, P_6, P_7, P_8\}$

CUADRO 3.2: Supervisores: RdP Ezpeleta (R).

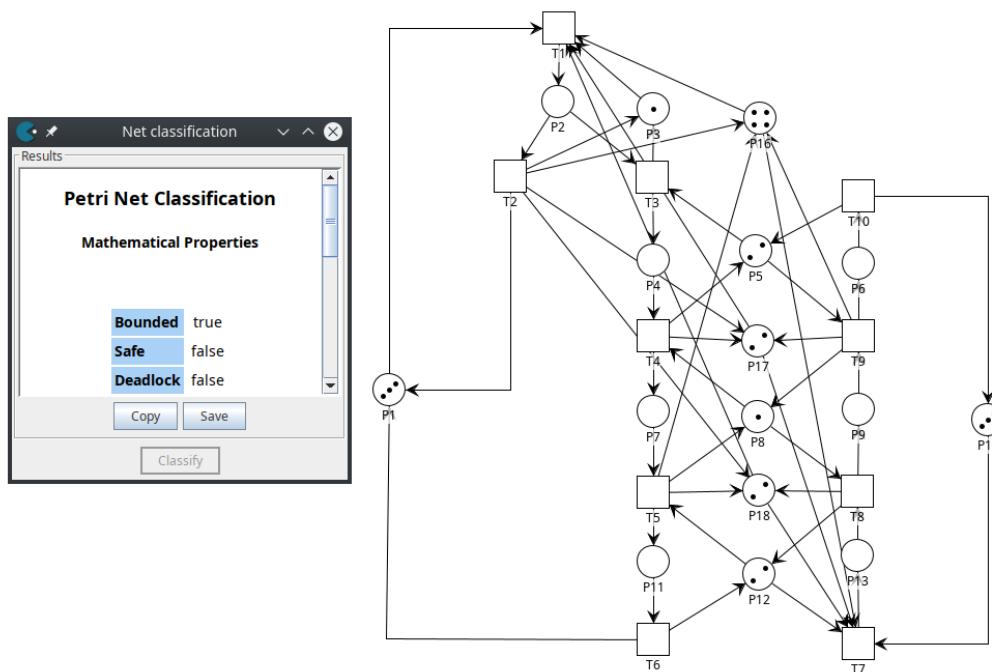


FIGURA 3.24: Subred derecha Ezpeleta controlada.

### Control red original

Al contemplar el conflicto en el análisis de cada subred por separado, la unión de estas soluciones permiten resolver el deadlock de la red completa, sin necesidad de realizar otro análisis.

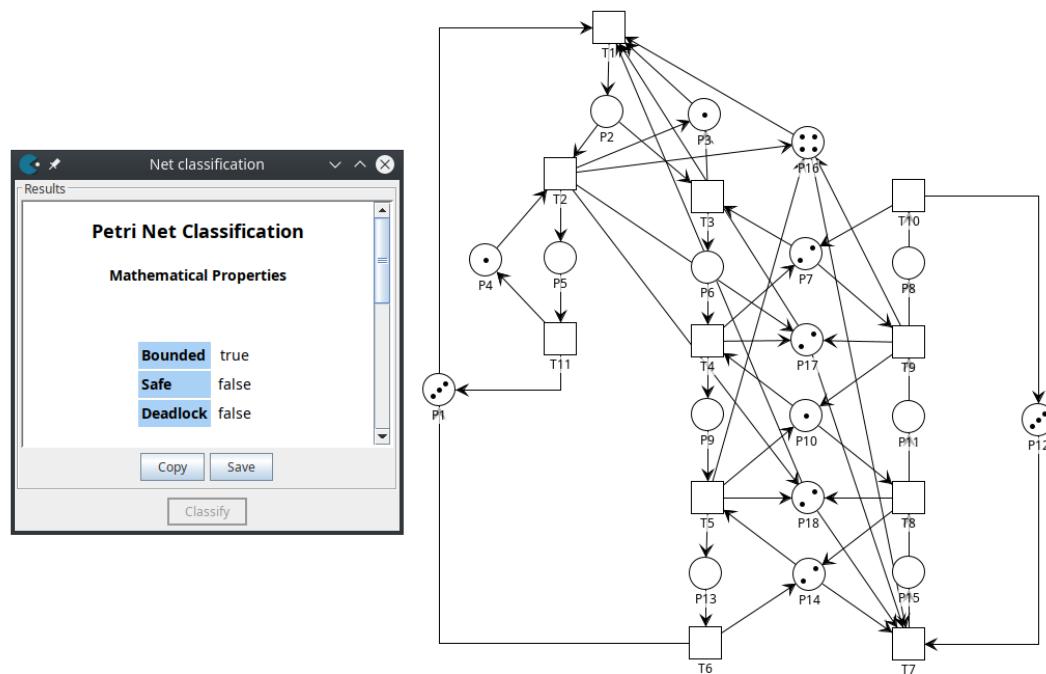


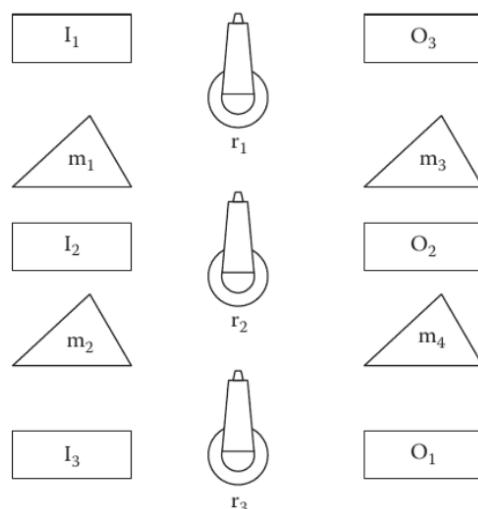
FIGURA 3.25: RdP Ezpeleta controlada.

### 3.5.6.3. Caso POPN

Esta red modela un sistema de manufacturación robotizado que consiste principalmente de: tres robots R1, R2, R3 y cuatro máquinas M1, M2, M3, M4.

En este sistema se procesan tres tipos diferentes de piezas A, B y C; estas provienen de tres contenedores de entrada distintos I1, I2 e I3, de los cuales los robots las retiran, las colocan en las máquinas para su procesamiento y posteriormente depositan en tres contenedores de salidas distintos, que son: O1, O2 y O3.

Todo esto se puede visualizar en Figura 3.26.

FIGURA 3.26: Modelado de partes del sistema POPN<sup>8</sup>.

<sup>8</sup>Figura adaptada del libro *System Modeling and Control with Resource-Oriented Petri Nets* [29].

Los robots tienen tareas definidas, cada operación de traslado de las piezas le corresponde a un único robot.

En la Figura 3.27, se observan las tres trayectorias diferentes para cada tipo de pieza.

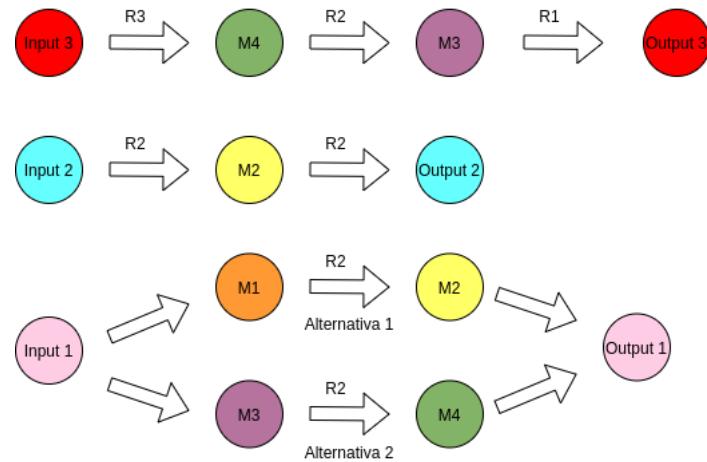


FIGURA 3.27: Trayectorias de la producción de las piezas.

Cada una de las máquinas, M1, M2, M3 y M4, tiene un color diferente. Y R1, R2 y R3 representan a los tres robots del sistema, encargados de trasladar las piezas de un lado a otro, hasta llegar a su objetivo. Si los recursos no se distribuyen de la forma correcta entre las máquinas la red se bloquea.

### 3.5.6.3.1. Características generales

- Presenta conflicto entre T-invariantes.
- Las máquinas M1-M4 están representadas por  $\{P_8, P_9, P_{21}, P_{22}\}$
- Mientras que los recursos R1-R3 están denotados por  $\{P_7, P_{14}, P_{10}\}$

### 3.5.6.3.2. Análisis estructural

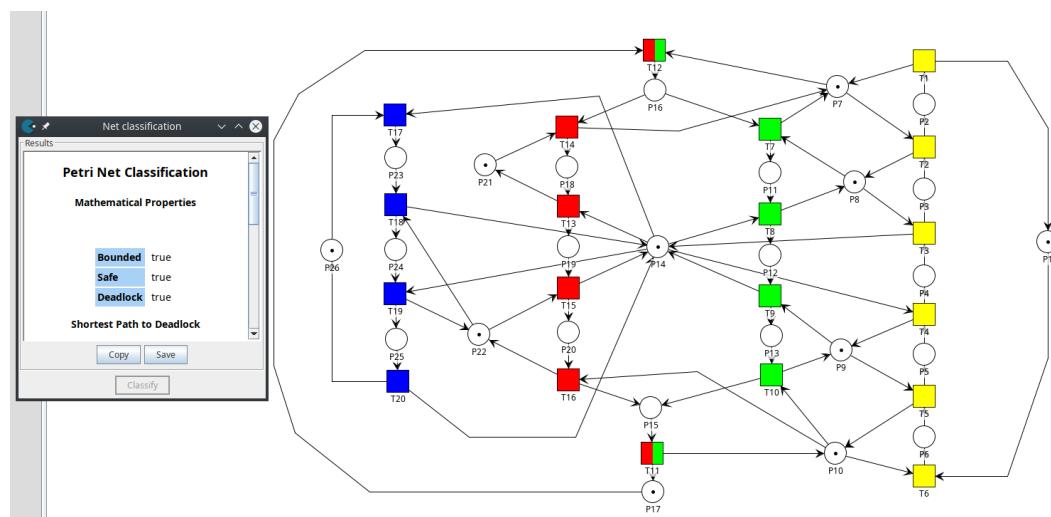


FIGURA 3.28: RdP POPN<sup>9</sup> y sus T-invariantes.

En la Figura 3.28, la plaza  $P_{16}$  forma parte de un conflicto, permitiendo la ejecución de un subcircuito de la red u otro, pudiendo seguir dos T-invariantes diferentes (rojo o verde en este caso). Es por esto que fue necesario ejecutar el algoritmo de forma completa, es decir desde el punto 1 al 5, contemplando la división de la red dado que los primeros 4 pasos no lograron alcanzar una red libre de deadlock. De la división resultaron dos subredes, preservando el conflicto (plaza y transiciones que lo conforman) en cada una de ellas.

### Subred derecha

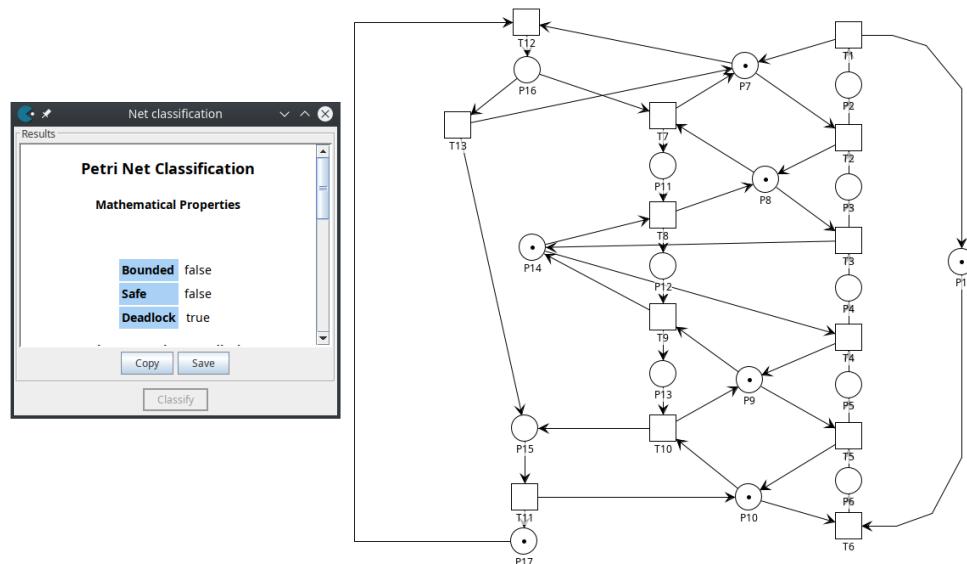


FIGURA 3.29: Subred derecha POPN.

### Subred izquierda

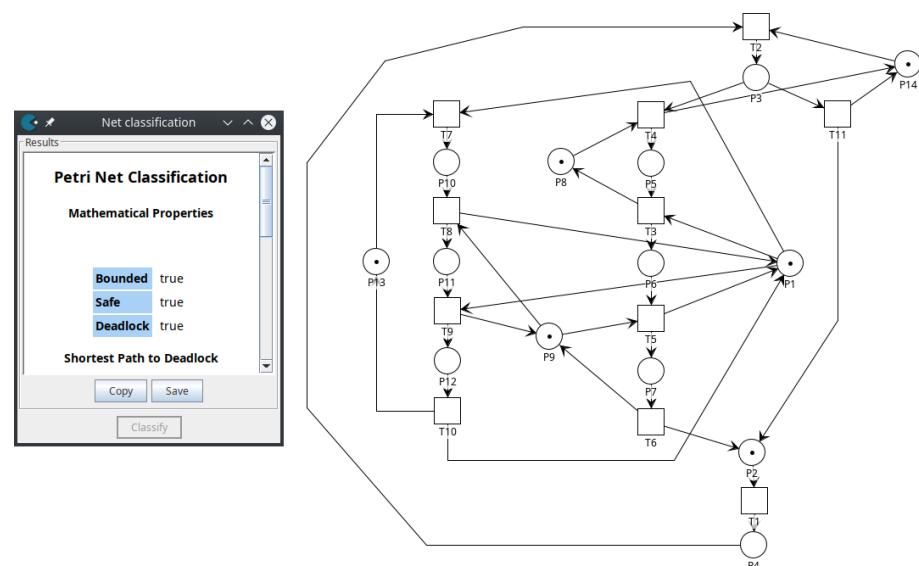


FIGURA 3.30: Subred izquierda POPN.

<sup>9</sup>Figura adaptada del libro *System Modeling and Control with Resource-Oriented Petri Nets* [29].

Dado que ambas subredes presentan deadlock es necesario ejecutar el algoritmo en cada una de ellas (desde el paso 1 al 4).

### Control de subredes

En las siguientes Figuras 3.31 y 3.32 se puede observar el control logrado en cada una de las subredes.

Supervisor	Marcado	Transiciones input	Transiciones output	Bad Siphon Controlado
$P_{28}$	1	$\{T_3, T_8, T_{14}\}$	$\{T_6, T_{12}\}$	$\{P_3, P_8, P_{12}, P_{14}\}$
$P_{29}$	1	$\{T_4, T_9, T_{14}\}$	$\{T_6, T_{12}\}$	$\{P_4, P_9, P_{13}, P_{14}\}$
$P_{30}$	1	$\{T_5, T_{10}, T_{14}\}$	$\{T_6, T_{12}\}$	$\{P_5, P_9, P_{10}, P_{15}\}$

CUADRO 3.3: Supervisores: RdP POPN (R).

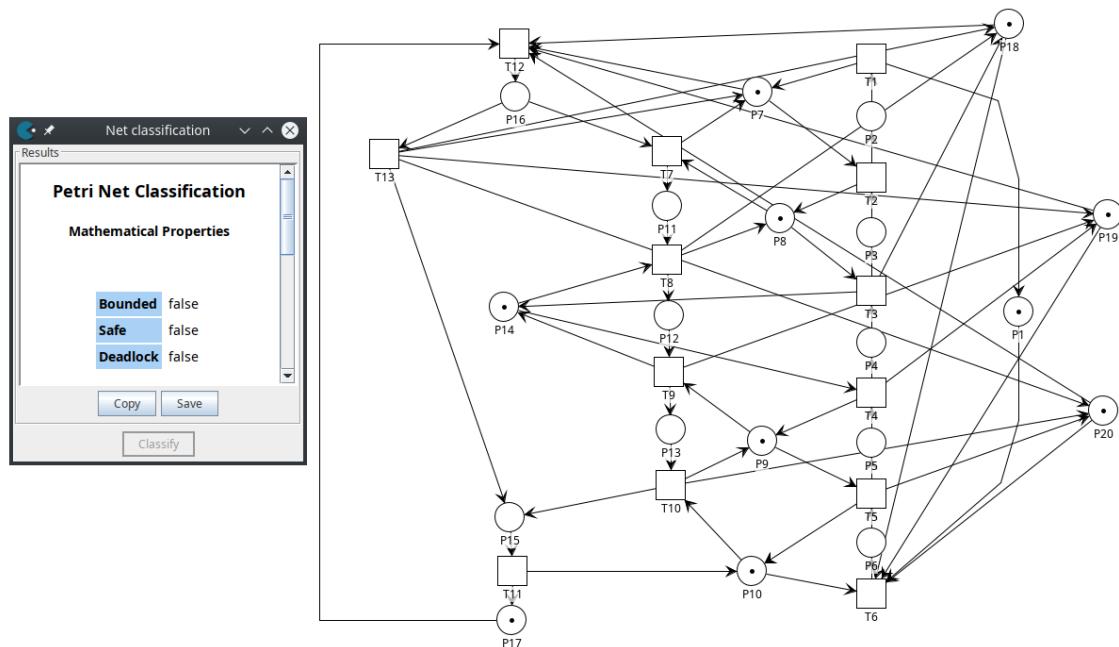


FIGURA 3.31: Subred derecha POPN controlada.

Supervisor	Marcado	Transiciones input	Transiciones output	Bad Siphon Controlado
$P_{27}$	1	$\{T_7, T_{15}, T_{19}\}$	$\{T_{12}, T_{17}\}$	$\{P_1, P_7, P_9, P_{12}\}$

CUADRO 3.4: Supervisores: RdP POPN (L).

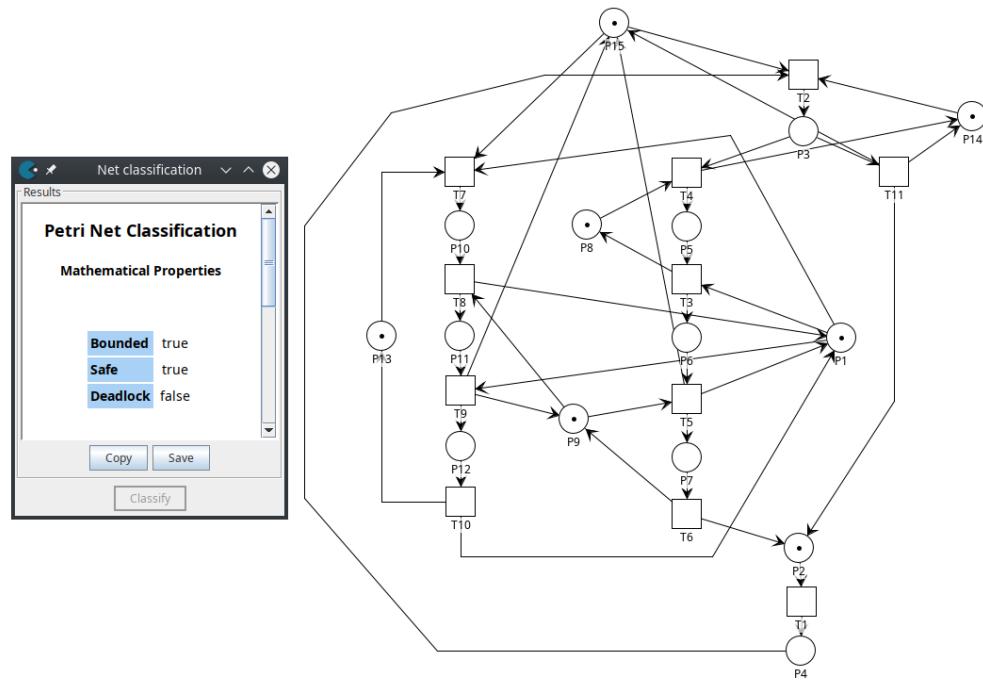


FIGURA 3.32: Subred izquierda POPN controlada.

### 3.5.6.3.3. Control red original

Al contemplar el conflicto en el análisis de cada subred, la unión de las soluciones individuales permiten resolver el deadlock de la red completa, sin necesidad de realizar otro análisis.

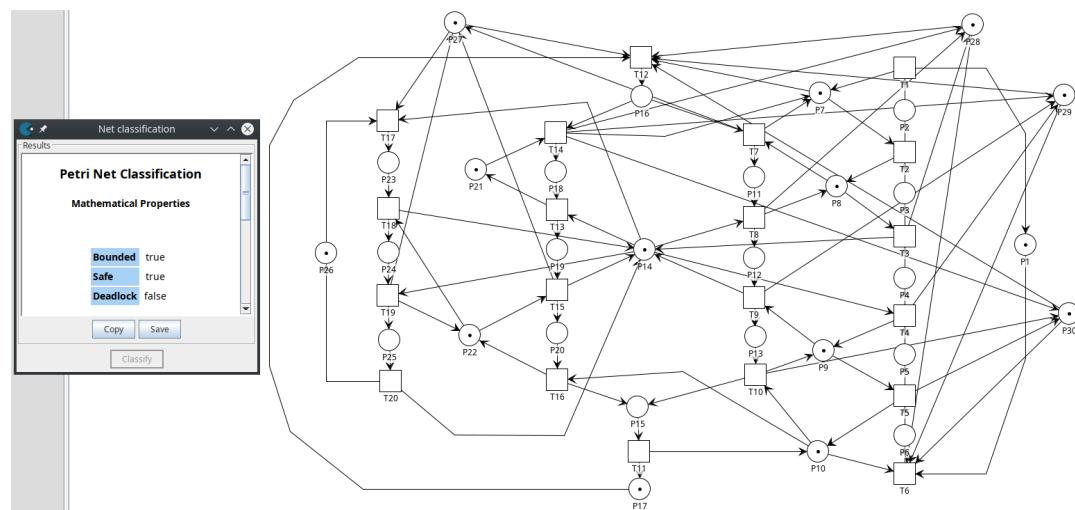


FIGURA 3.33: RdP POPN controlada.

#### 3.5.6.4. Caso Guanjun

Esta red modela la ejecución concurrente de procesos de trabajo en FMS que describe el comportamiento de 2 subredes relacionadas por dos recursos, en caso de llevar una mala gestión de estos la red terminará en deadlock.

#### **3.5.6.4.1. Características generales**

- Presenta conflicto entre T-invariantes.
  - Los recursos R1-R5 están representados por las plazas  $\{P_{11}, P_{12}, P_{14}, P_{15}, P_{18}\}$ , con  $P_{14}$  y  $P_{15}$  compartidas por las dos subredes.

## Análisis estructural

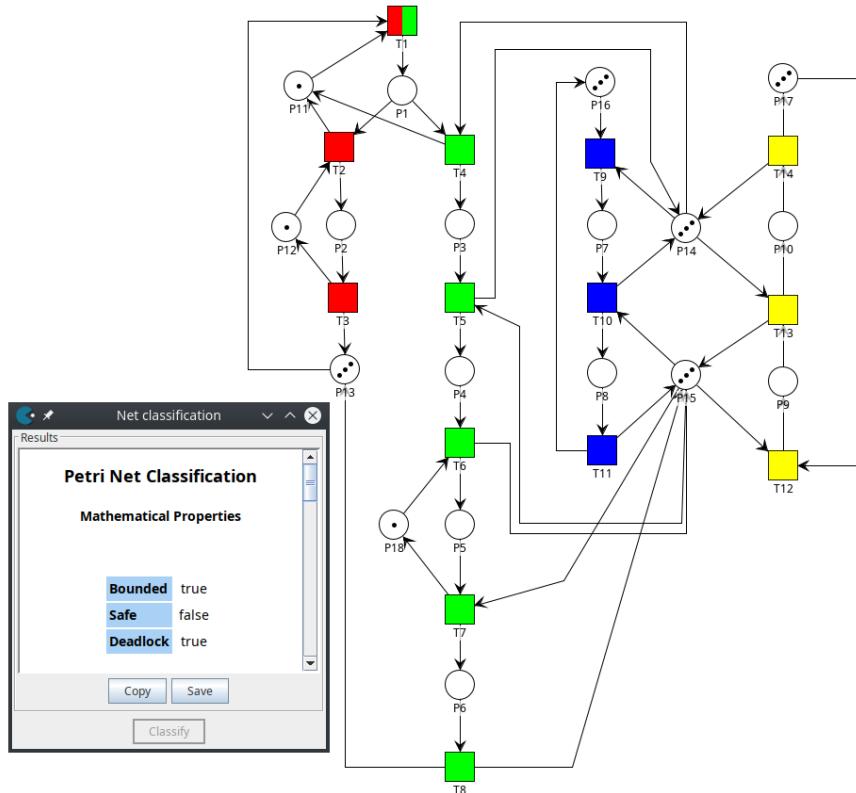


FIGURA 3.34: RdP Guanjun<sup>10</sup> y sus T-invariantes.

En la Figura 3.34, la plaza  $P_1$  forma parte de un conflicto permitiendo la ejecución de un subcircuito de la red u otro, pudiendo seguir dos T-invariantes diferentes (rojo o verde en este caso). Pero en este caso, a diferencia de los anteriores, al ejecutar el algoritmo mencionado desde el punto 1-4 por primera vez sobre la red completa, permite encontrar los supervisores que resuelven el deadlock sin necesidad de subdividir la red (es decir, ejecutar el paso 5). Dado que el token independientemente del T-invariante que siga siempre vuelve a los supervisores, como se puede observar en la Figura 3.35.

<sup>10</sup>Figura adaptada del paper publicado por G. Liu, C. Jiang and M. Zhou [5].

### Red controlada

Una vez ejecutado el algoritmo, se obtienen la plaza y los arcos a agregar para controlar la red, solucionando el problema de deadlock.

Supervisor	Marcado	Transiciones input	Transiciones output	Bad Siphon Controlado
$P_{19}$	6	$\{T_2, T_7, T_{10}, T_{13}\}$	$\{T_1, T_9, T_{12}\}$	$\{P_6, P_8, P_{10}, P_{14}, P_{15}, P_{18}\}$
$P_{20}$	5	$\{T_2, T_5, T_{10}, T_{13}\}$	$\{T_1, T_9, T_{12}\}$	$\{P_4, P_6, P_8, P_{10}, P_{14}, P_{15}\}$

CUADRO 3.5: Supervisores: RdP Guanjun.

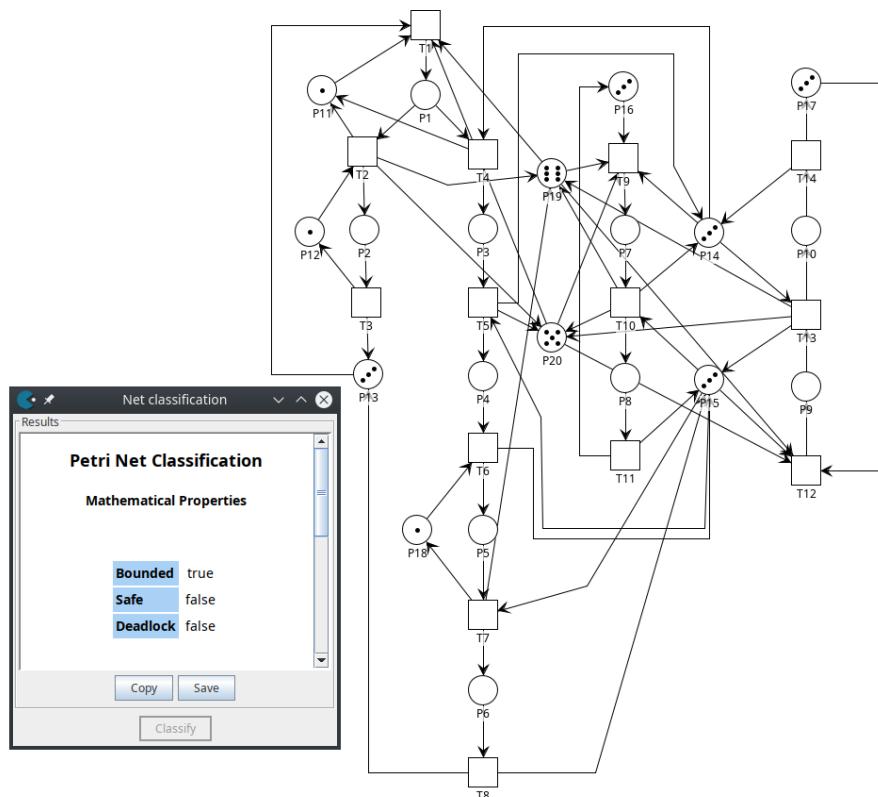


FIGURA 3.35: RdP Guanjun controlada.

### 3.5.7. Conclusión

Para lograr el correcto funcionamiento del algoritmo fue de vital importancia incorporar un tercer arco, el cual Ezpeleta en su desarrollo omitía para las situaciones en donde la red presentaba conflicto entre sus T-invariantes. Es de vital importancia dado que si este no estuviera presente, el supervisor alcanzaría un marcado cero producto del disparo de la transición que en su trayectoria no contempla ningún arco que devuelva el token al supervisor y en consecuencia la red no volvería a ejecutarse.



## Capítulo 4

# Testing

Una vez analizados todos los casos de estudios con sus respectivas particularidades, se logró un algoritmo conforme con las especificaciones necesarias antes definidas. Una vez alcanzado el mismo se decidió poner a prueba su desempeño ante nuevos escenarios.

Para cada uno de estos nuevos escenarios se llevó a cabo el mismo análisis realizado para los casos de estudio anteriores, sin embargo para no ser reiterativos en este aspecto no se especifican la totalidad de los pasos hasta resolver el deadlock, dado que ya han sido mencionados en el desarrollo de la investigación.

### 4.1. Nuevos escenarios

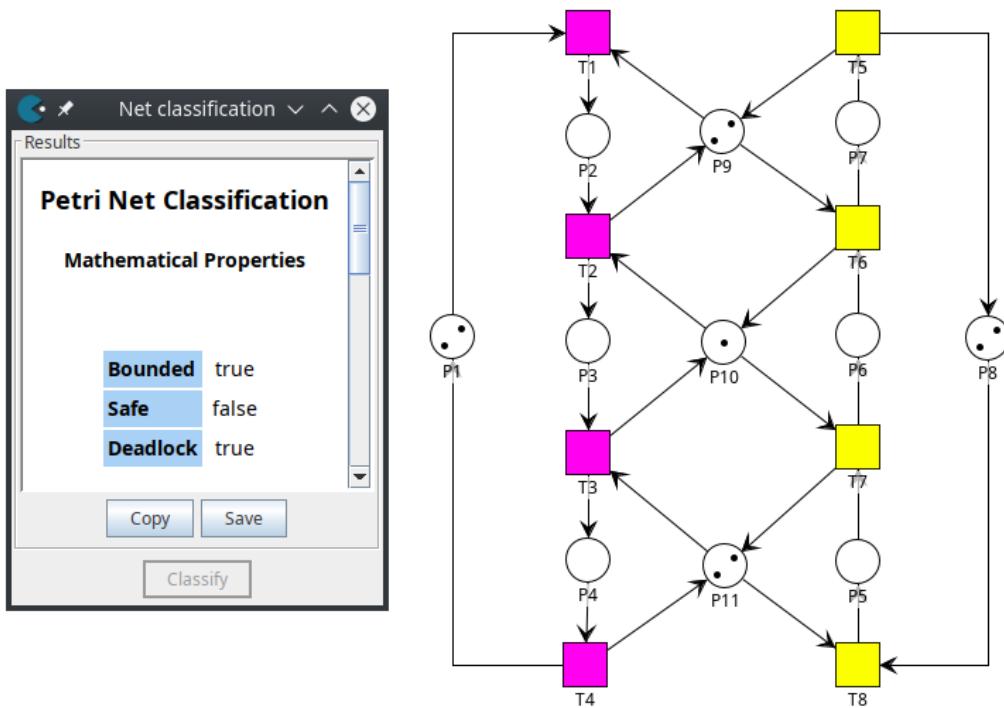
#### 4.1.1. Caso Hiuxia

Esta red modela la ejecución concurrente de un AMS, este tipo de sistemas consiste en un conjunto de recursos finitos como máquinas, búferes y robots. Diferentes tipos de piezas ingresan al sistema en momentos discretos y se procesan simultáneamente. Todas las partes procesadas en el sistema compiten por estos recursos finitos; por lo tanto, pueden producirse problemas como bloqueos, conflictos e interbloqueos.

##### 4.1.1.1. Características generales red Hiuxia

- Los recursos de la red están representados por las plazas  $\{P_9, P_{10}, P_{11}\}$ .

##### 4.1.1.2. Análisis estructural red Hiuxia

FIGURA 4.1: RdP Hiuxia<sup>1</sup> y sus T-invariantes.

En la figura 4.1, se presentan los T-invariantes en diferentes colores.

#### 4.1.1.2.1. Red controlada

Una vez ejecutado el algoritmo, se obtienen los supervisores a agregar para controlar la red solucionando el deadlock, estos están representado por las plazas  $P_{12}$  y  $P_{13}$  en la Figura 4.2.

Supervisor	Marcado	Transiciones input	Transiciones output	Bad Siphon Controlado
$P_{12}$	2	$\{T_2, T_6\}$	$\{T_1, T_8\}$	$\{P_3, P_7, P_9, P_{10}\}$
$P_{13}$	2	$\{T_3, T_7\}$	$\{T_1, T_8\}$	$\{P_4, P_6, P_{10}, P_{11}\}$

CUADRO 4.1: Supervisores: RdP Hiuxia

<sup>1</sup>Figura adaptada del paper publicado por Huixia Liu et al. [15]

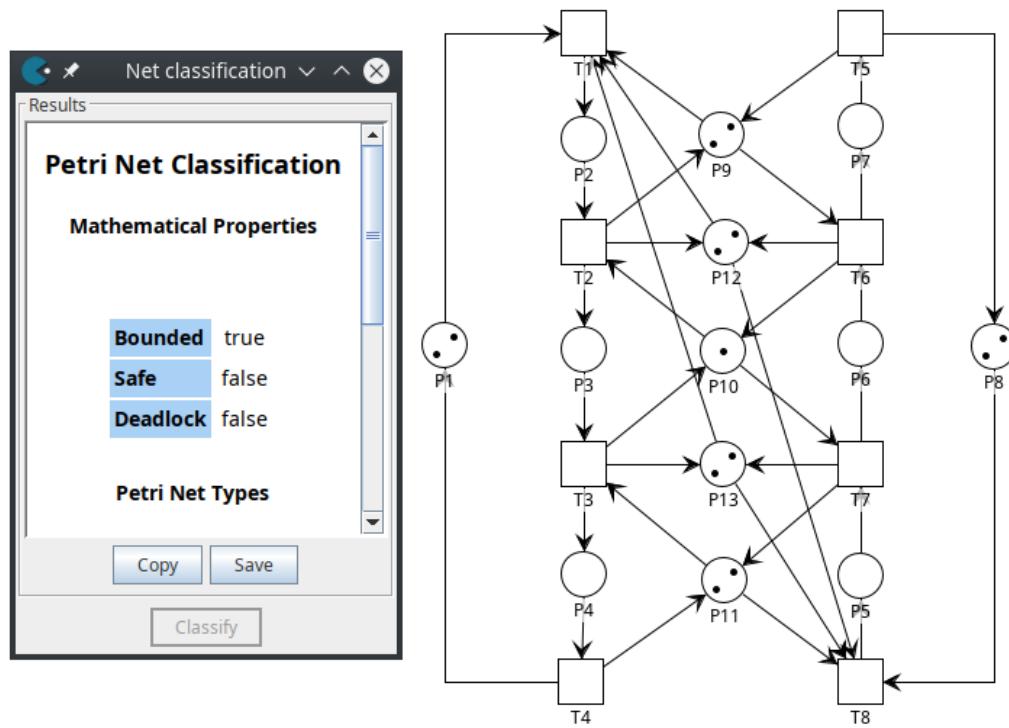


FIGURA 4.2: RdP Hiuxia controlada.

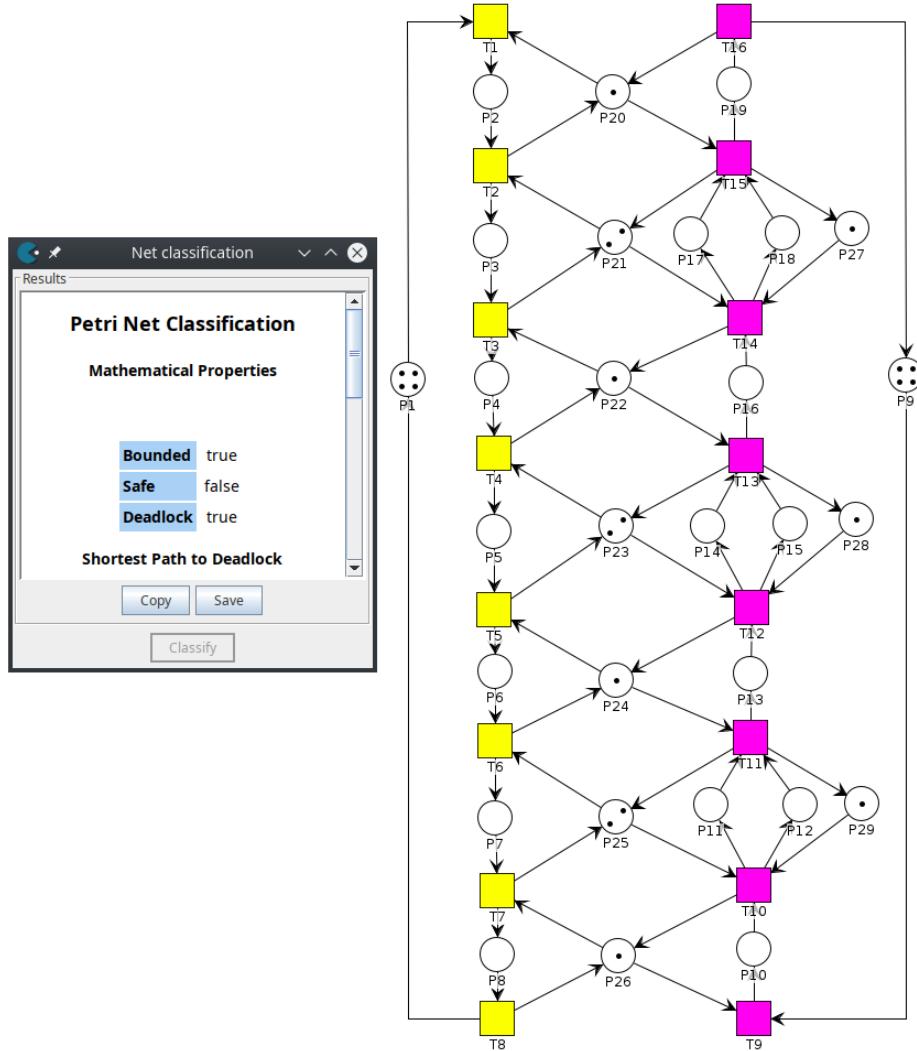
#### 4.1.2. Caso JianChao

Esta red modela la ejecución concurrente de un AMS, se presentan dos procesos productivos que comparten compuesto por 4 robot y 6 maquinas. Un control incorrecto de estos en la ejecución de los procesos de trabajo, puede conducir a situaciones de deadlock.

##### 4.1.2.1. Características generales

- Los robot de la red están representados por las plazas  $\{P_{20}, P_{22}, P_{24}, P_{26}\}$ .
- Las maquinas de la red están representados por las plazas  $\{P_{21}, P_{23}, P_{25}, P_{27}, P_{28}, P_{29}\}$ .

##### 4.1.2.2. Análisis estructural

FIGURA 4.3: RdP JianChao<sup>2</sup> y sus T-invariantes.

En la figura 4.3<sup>3</sup>, se presentan los T-invariantes en diferentes colores.

#### 4.1.2.2.1. Red controlada

Una vez ejecutado el algoritmo, se obtienen los supervisores a agregar para controlar la red solucionando el deadlock, estos están representado por las plazas  $P_{30}$ ,  $P_{31}$  y  $P_{32}$  en la Figura 4.4.

Supervisor	Marcado	Transiciones input	Transiciones output	Bad Siphon Controlado
$P_{30}$	2	$\{T_7, T_{10}\}$	$\{T_1, T_9\}$	$\{P_8, P_{11}, P_{25}, P_{26}\}$
$P_{31}$	2	$\{T_5, T_{12}\}$	$\{T_1, T_9\}$	$\{P_6, P_{14}, P_{23}, P_{24}\}$
$P_{32}$	2	$\{T_3, T_{14}\}$	$\{T_1, T_9\}$	$\{P_4, P_{17}, P_{21}, P_{22}\}$

CUADRO 4.2: Supervisores: RdP JianChao

<sup>2</sup>Figura adaptada del paper publicado por JianChao Lou et al. [16]

<sup>3</sup>En esta RdP fue modificado el peso de los arcos reduciéndolos a uno para adaptarlas al tipo de red que admite el algoritmo.

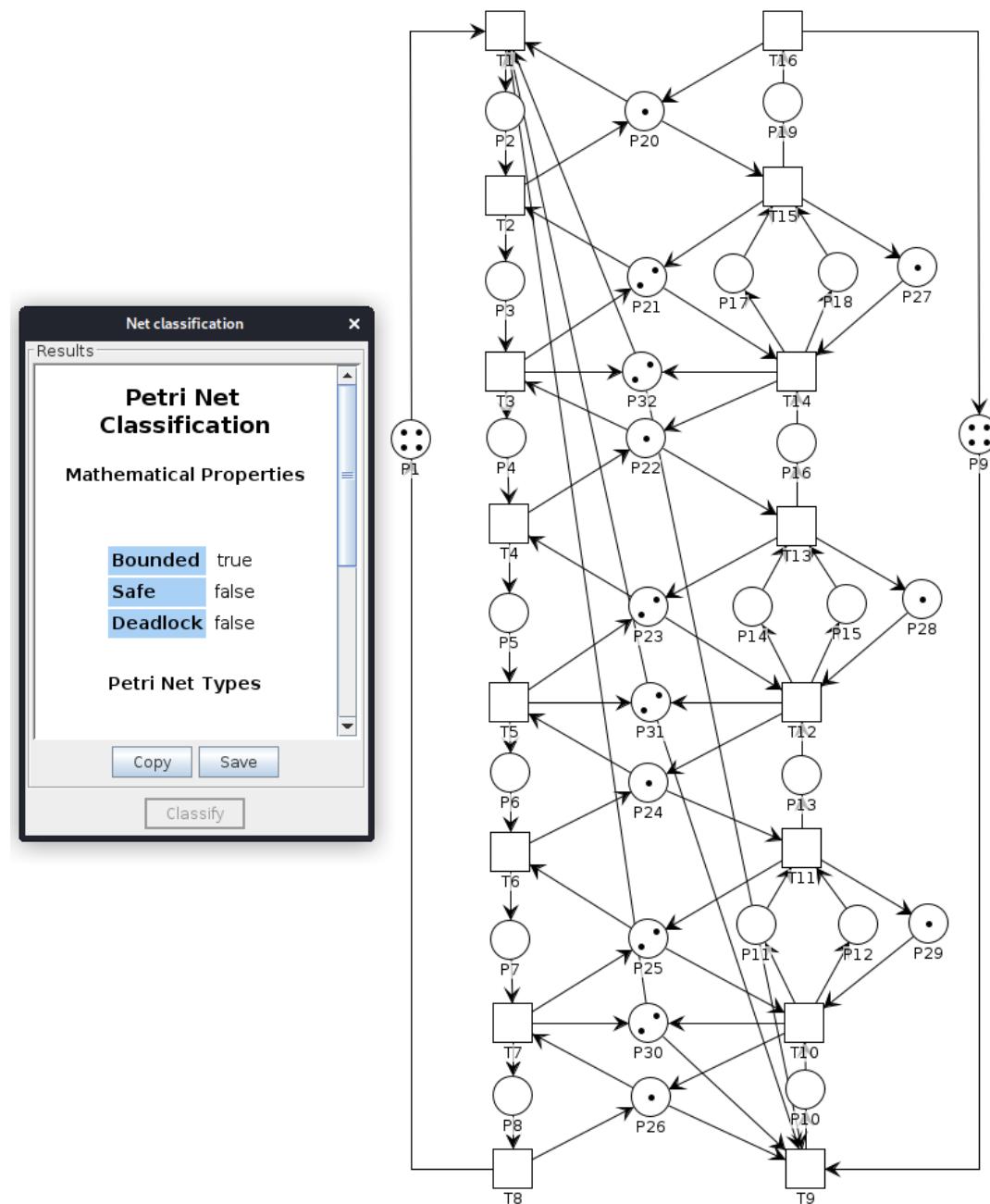


FIGURA 4.4: RdP JianChao controlada.

#### 4.1.2.3. Caso Zhao

Esta red modela la ejecución concurrente de procesos de trabajo en FMS que describe el comportamiento de 3 subredes relacionadas por tres recursos, en caso de llevar una mala gestión de estos la red terminará en deadlock.

##### 4.1.2.3.1. Características generales

- Presenta conflictos entre T-invariantes.
- Los recursos R1-R3 están representados por las plazas  $\{P_{15}, P_{16}, P_{17}\}$ .

## Análisis estructural

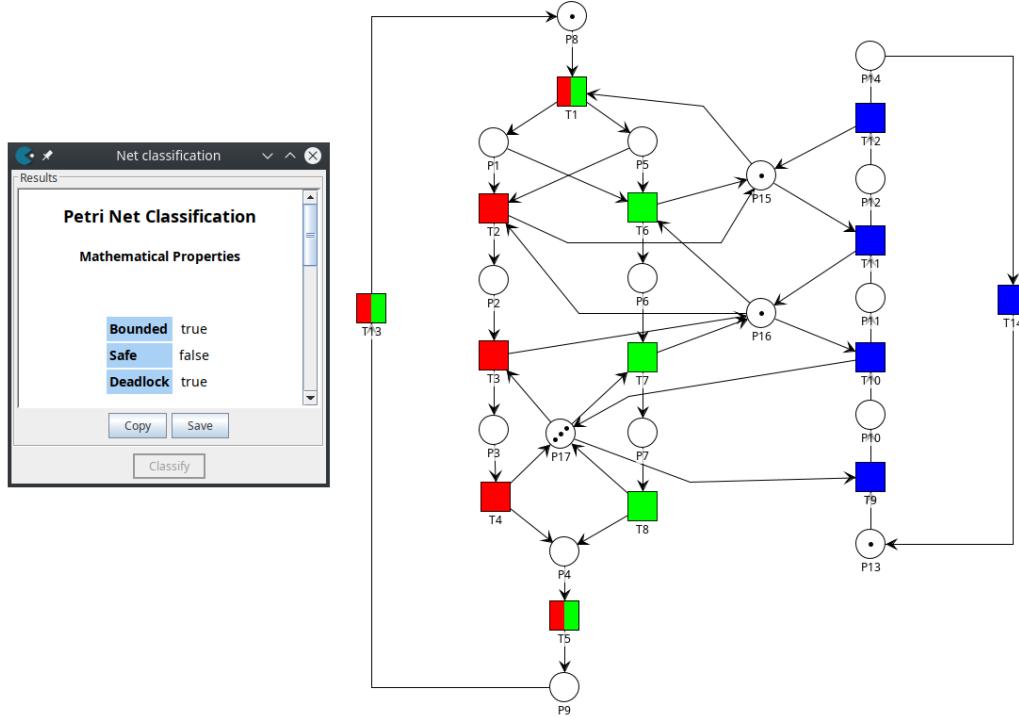


FIGURA 4.5: RdP Zhao<sup>4</sup> y sus T-invariantes.

En la figura 4.5<sup>5</sup>, las plazas  $P_1$  y  $P_5$  forma parte (cada una) de un conflicto permitiendo su disparo la ejecución de un subcircuito de la red u otro, pudiendo seguir dos T-invariantes diferentes (rojo o verde en este caso). Pero en este caso, al igual que en la red Guanjun (Sección 3.5.6.4), al ejecutar el algoritmo mencionado desde el punto 1-4 por primera vez sobre la red completa, permite encontrar los supervisores que resuelven el deadlock sin necesidad de subdividirla (es decir, ejecutar el paso 5). Dado que el token independientemente del T-invariante que siga siempre vuelve a los supervisores, como se puede observar en la figura 4.6.

### Red controlada

Una vez ejecutado el algoritmo e incorporando el supervisor determinado (plaza y arcos correspondientes) a la red, se logra controlar el problema de deadlock.

Supervisor	Marcado	Transiciones input	Transiciones output	Bad Siphon Controlado
$P_{18}$	1	$\{T_2, T_6, T_{11}\}$	$\{T_1, T_9\}$	$\{P_2, P_6, P_{12}, P_{15}, P_{16}\}$

CUADRO 4.3: Supervisores: RdP Zhao.

<sup>4</sup>Figura adaptada del paper publicado por Mi Zhao et al. [28].

<sup>5</sup>En esta RdP fue modificado el peso de los arcos reduciéndolos a uno (1) para adaptarla al tipo de red que admite el algoritmo.

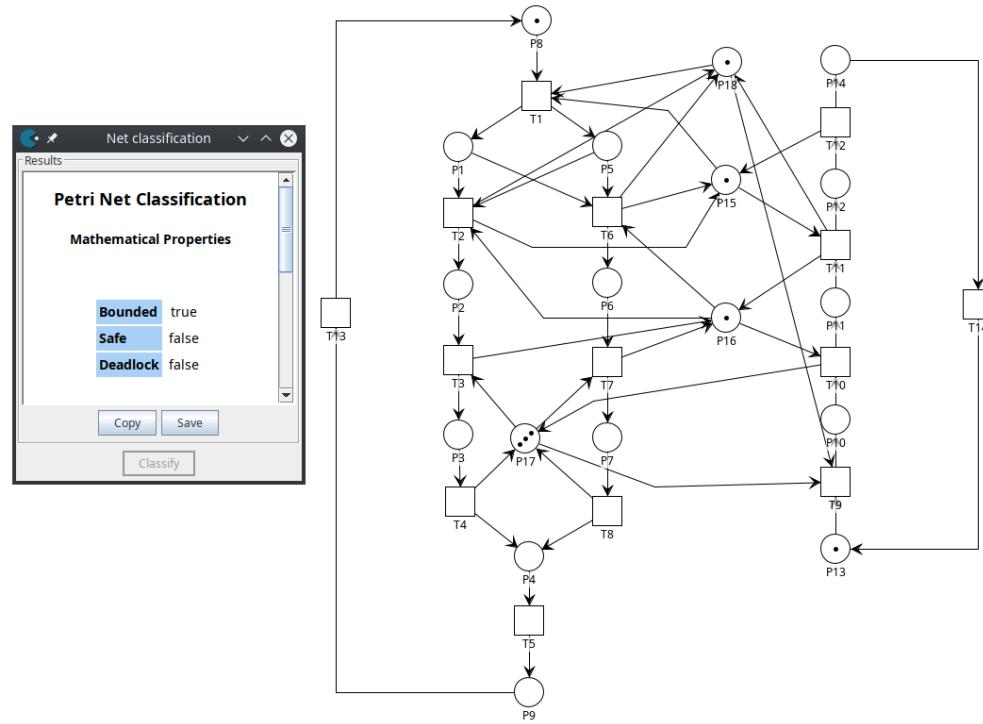


FIGURA 4.6: RdP Zhao controlada.

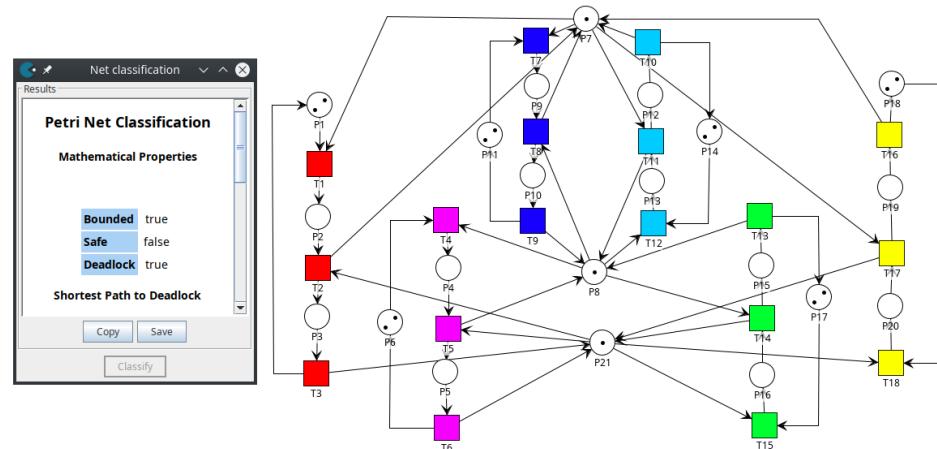
### 4.1.3. Caso Auto

Esta red modela la ejecución concurrente de un AMS, donde existen plazas que simulan la disponibilidad de recursos (3) y un control incorrecto de estos en la ejecución de los procesos de trabajo, puede conducir a situaciones de deadlock.

#### 4.1.3.1. Características generales red Auto

- Los recursos de la red están representados por las plazas  $\{P_{19}, P_{20}, P_{21}\}$ .

#### 4.1.3.2. Análisis estructural Auto

FIGURA 4.7: RdP Auto<sup>6</sup> y sus T-invariantes.

En la figura 4.7, se presentan los T-invariantes en diferentes colores.

#### 4.1.3.2.1. Control de la red

Al agregar el primer supervisor en la red ésta aún presentaba deadlock, cuando se ejecutó nuevamente el algoritmo en busca de otro supervisor el marcado que daba este era de cero. Se intento reiniciar la ejecución del algoritmo tomando como punto de partida otro supervisor inicial y en todos los casos se obtuvo el mismo resultado, es decir, no se llegó a controlar el deadlock de la red.

A diferencia de otras redes en las que el análisis global resultó fallido y luego se realizaba el análisis de la red por las subredes que la componían; en este caso no se pudo realizar la misma estrategia dado que no se encontraron transiciones en conflicto como para llevar a cabo la división.

#### Primera ejecución

Como se mencionó con anterioridad se colocó el primer supervisor definido en el Cuadro 4.4. En la Figura 4.8 la red aún presentaba deadlock.

Supervisor	Marcado	Transiciones input	Transiciones output	Bad Siphon Controlado
$P_{22}$	1	$\{T_8, T_{11}\}$	$\{T_1, T_4, T_7, T_{12}, T_{15}, T_{18}\}$	$\{P_2, P_5, P_9, P_{10}, P_{13}, P_{17}, P_{19}, P_{20}\}$

CUADRO 4.4: Supervisores: RdP Auto - Primera ejecución

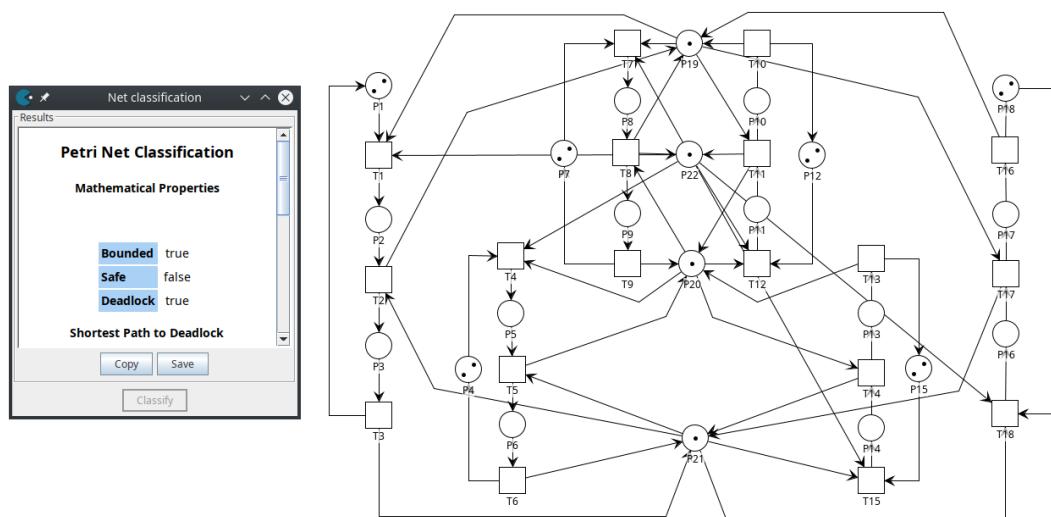


FIGURA 4.8: RdP Auto sin controlar.

Al exportar los archivos de la red y ejecutando nuevamente el algoritmo se puede observar en la siguiente figura que el siguiente supervisor a colocar posee marcado cero. Este era uno de los impedimentos mencionados Sección 3.5.4.1 indicando que este no era una solución óptima dado que imposibilita la ejecución de una parte de la red y la idea es lograr el control de la red manteniendo el comportamiento original de la misma.

<sup>6</sup>Figura adaptada del paper publicado por Huixia Liu et al. [15]

```

Path del archivo de Estados (html): g2.html
Path del archivo de Matrices I(html): m2.html
Path del archivo de Sifones(html): s2.html
Path del archivo de Invariantes (html): i2.html
Sifones vacios en idle []
Estados con deadlock [3]
Cantidad de estados con deadlock 1
Estado deadlock, sifon asociado al deadlock y su marcado [[3, 7, 1]]
Cantidad de sifones vacios: 1

Sifon a controlar: 7
Marcado del supervisor 0 ←
Transicion output: 1
Transicion output: 4
Transicion output: 7
Transicion output: 12
Transicion output: 15
Transicion output: 18
sifones vacios sin repetir [7]
Cantidad de sifones vacios sin repetir 1

```

FIGURA 4.9: Supervisor con marcado cero.

### Segunda ejecución

En esta segunda ejecución se buscó otro enfoque al momento de la elección del supervisor, colocando otro de los sugeridos. Y al igual que en la ejecución anterior el marcado del segundo supervisor a colocar era cero, como se muestra en la Figura 4.9.

Supervisor	Marcado	Transiciones input	Transiciones output	Bad Siphon Controlado
$P_{22}$	1	$\{T_2, T_{17}\}$	$\{T_1, T_4, T_7, T_{12}, T_{15}, T_{18}\}$	$\{P_3, P_6, P_8, P_{10}, P_{14}, P_{17}, P_{19}, P_{21}\}$

CUADRO 4.5: Supervisores: RdP Auto - Segunda ejecución

### Tercera ejecución

En esta tercera ejecución se logró colocar hasta dos supervisores pero al momento de colocar el tercero, al igual que en las ejecuciones anteriores, el marcado era cero y no se logró resolver el estado de deadlock, como se muestra en la Figura 4.9.

Supervisor	Marcado	Transiciones input	Transiciones output	Bad Siphon Controlado
$P_{22}$	1	$\{T_2, T_5, T_8, T_{11}, T_{14}, T_{17}\}$	$\{T_1, T_4, T_7, T_{12}, T_{15}, T_{18}\}$	$\{P_3, P_6, P_9, P_{10}, P_{13}, P_{17}, P_{19}, P_{20}, P_{21}\}$
$P_{23}$	1	$\{T_5, T_{14}\}$	$\{T_1, T_4, T_7, T_{12}, T_{15}, T_{18}\}$	$\{P_3, P_6, P_9, P_{11}, P_{13}, P_{16}, P_{20}, P_{21}\}$

CUADRO 4.6: Supervisores: RdP Auto - Tercera ejecución



## Capítulo 5

# Conclusión

En el capítulo introductorio del presente documento se especificaron los objetivos planteados para el proyecto desarrollado. Una vez finalizadas las tareas asociadas a los mismos, es posible extraer algunas conclusiones.

Por un lado se logró establecer una interfaz entre nuestro algoritmo y el software Petrinator, como ya se mencionó con anterioridad, en lo que sería la extracción manual de los archivos y su posterior procesamiento.

Durante el desarrollo de este proyecto se planteo la estrategia de utilizar los dos enfoques mencionados en la Sección 1.2 sacando provecho tanto de las propiedades estructurales de la red como el estudio del grafo de alcanzabilidad y de esta manera explotar sus ventajas y contrarrestar sus limitaciones.

En cuanto a la enfoque estructural se detectó la necesidad de extraer los T-invariantes de la red, dado que estos debían preservarse a lo largo del análisis ya que la idea es controlar la red manteniendo su comportamiento original.

Un punto importante a destacar es que en ninguno de los papers referenciados realiza un enfoque en la utilización de esta propiedad estructural para lograr la solución de los estados de deadlock.

Mientras que el enfoque haciendo uso del grafo de alcanzabilidad nos permitió obtener los estados de deadlock y a partir de los mismos identificar cuales eran los sifones mínimos vacíos. Cabe destacar que el tamaño del grafo de alcanzabilidad en la implementación de redes de mayor número de estados la ejecución se vio limitada por las características del ordenador en el que se ejecutó.

A partir de lo mencionado y demostrado a lo largo de la evolución del algoritmo, este se fue probando en redes con estructuras diferentes (conflictos presentes en la red, la distribución/relación entre los bad siphons y los T-invariantes) que permitió demostrar fiabilidad del mismo como también agregarle nuevas características para llegar al objetivo planteado.

La generación de código a partir del control de los estados de deadlock en las redes de Petri S<sup>3</sup>PR, simulando el comportamiento que llevaría a cabo un monitor, era uno de los objetivos. El mismo se alcanzó de manera parcial dado que se encontró el caso (Sección 4.1.3.2) en el que la vivacidad de la red no se llegó a alcanzar.

De esta forma el algoritmo puede ser tomado como base para el estudio y posterior desarrollo en el área de control de redes de Petri.

## **5.1. Trabajo a futuro**

A continuación se listan los posibles avances que se podría realizar para continuar este proyecto:

- Integrar este algoritmo como una nueva funcionalidad del software Petrinator permitiendo la ejecución automática del mismo sin necesidad de la extracción manual de archivos.
- Posibilidad de integrar este proyecto junto con los que también se están desarrollando en el LAC con la idea de lograr tanto el control de la red como la ejecución de la misma (tanto en hardware como en software) a partir de ciertas políticas definidas.
- Generar una nueva interfaz que conecte el algoritmo con el software Petrinator permitiendo automatizar la colocación de los supervisores.
- Realizar un generador aleatorio de redes de Petri que sirvan de entrada para un testeo más amplio del algoritmo.
- Se propone seguir investigando y testeando con nuevos tipos de redes bloqueadas que presenten diferentes características a las ya analizadas.

## Apéndice A

# Tutorial de como utilizar el software *Petrinator*

En este apéndice, se detalla el procedimiento en el cual se crean y cargan redes de Petri para trabajar con ellas con dicho software. Luego se explica el procedimiento que se debe seguir para extraer los datos necesarios para el análisis del algoritmo.

### A.1. Ejecución del software

A continuación explican los pasos que se deben seguir para poder iniciar el agente *Petrinator* en el dispositivo.

- **Ingreso al programa:** desde la terminal de linux<sup>1</sup> se ejecuta el comando:

```
$ java -jar Petrinator.jar
```

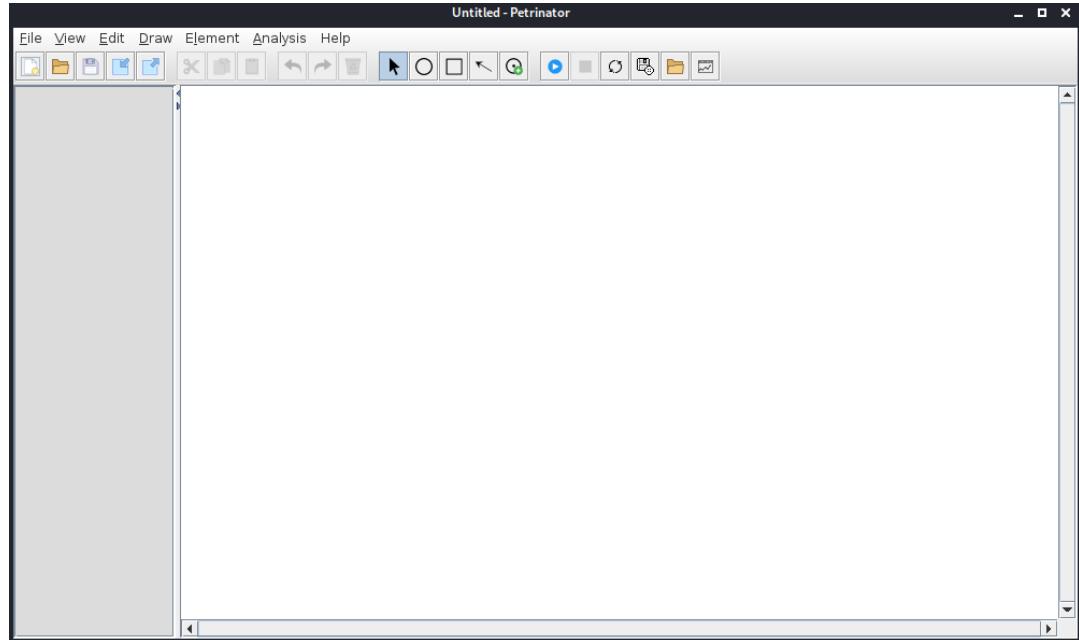


FIGURA A.1: Software Petrinator .

- **Cargar la red de Petri:** una vez ejecutado exitosamente, se deberá importar o crear la red a analizar. Como se muestra en la figura A.3.

---

<sup>1</sup>La ejecución del software tambien puede realizarse en Windows. Para ambos casos es necesario instalar los paquetes correspondientes de java.

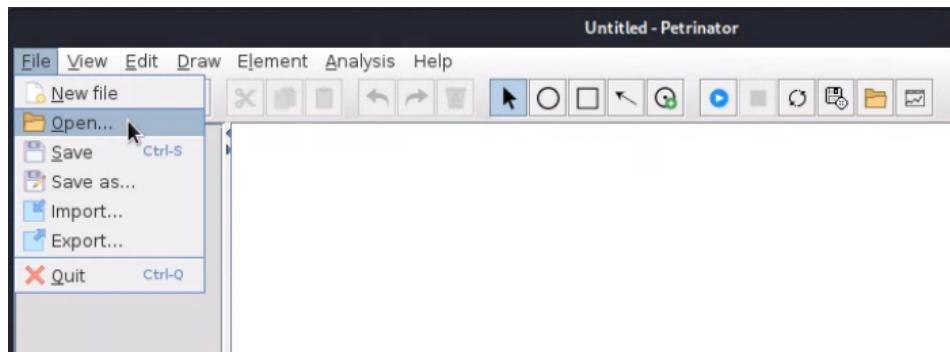


FIGURA A.2: Abrir un archivo .

Seleccionar el archivo de extensión .pflow a cargar

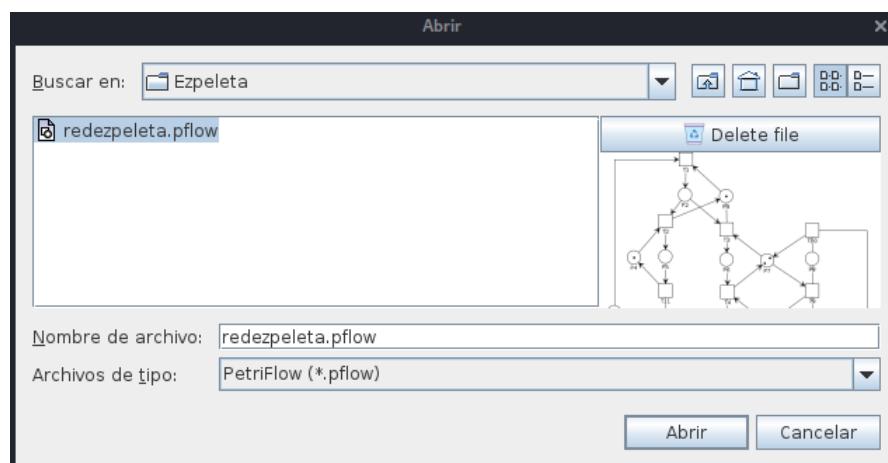


FIGURA A.3: Ventana de selección de archivo .pflow .

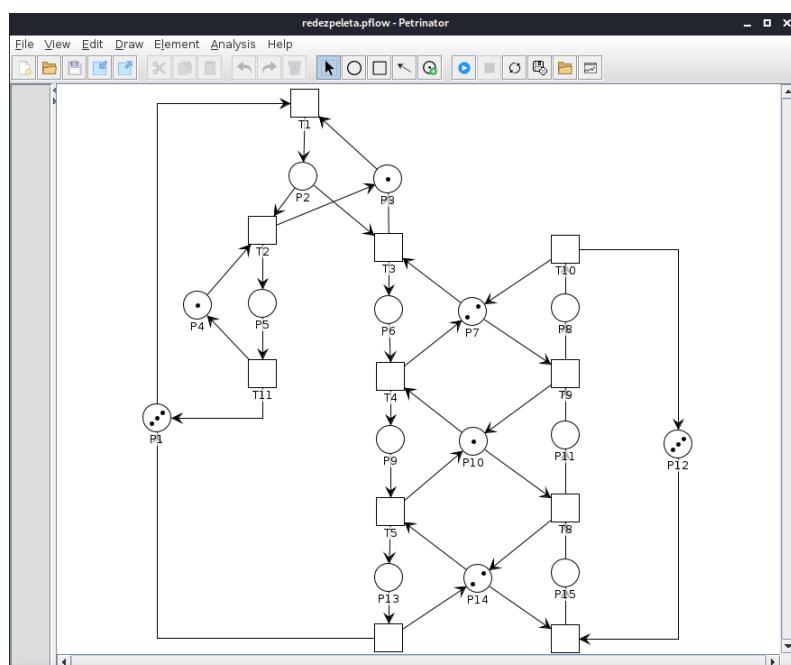


FIGURA A.4: Red cargada .

- **Extraer archivos para el análisis:** Como se puede observar en la figura A.5, en la barra de menú se encuentra la opción de *Analysis* desde donde se pueden realizar los diferentes tipos de análisis en las redes de Petri. A partir de estos, se extraen 4 diferentes archivos.

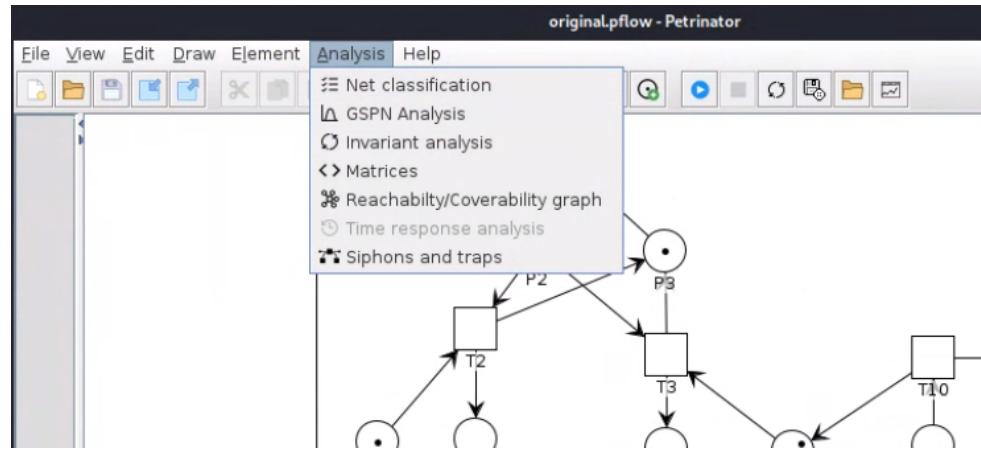


FIGURA A.5: Submenú de análisis.

- **Clasificación de la red:** como se observa en la figura A.5 se encuentra en el submenú la opción *Net classification* que nos permite realizar este análisis. Una vez seleccionada la opción, se abre una nueva ventana como muestra la figura A.6 en dónde se debe pulsar en *Classify* para que el software haga dicho análisis. Para la funcionalidad de nuestro algoritmo no es necesario exportar este archivo, sin embargo es de imprescindible verificar la presencia de Deadlock. En caso de haber bloqueo es necesaria la extracción de los siguientes archivos para la ejecución del algoritmo; en caso contrario no sería una red de interés.

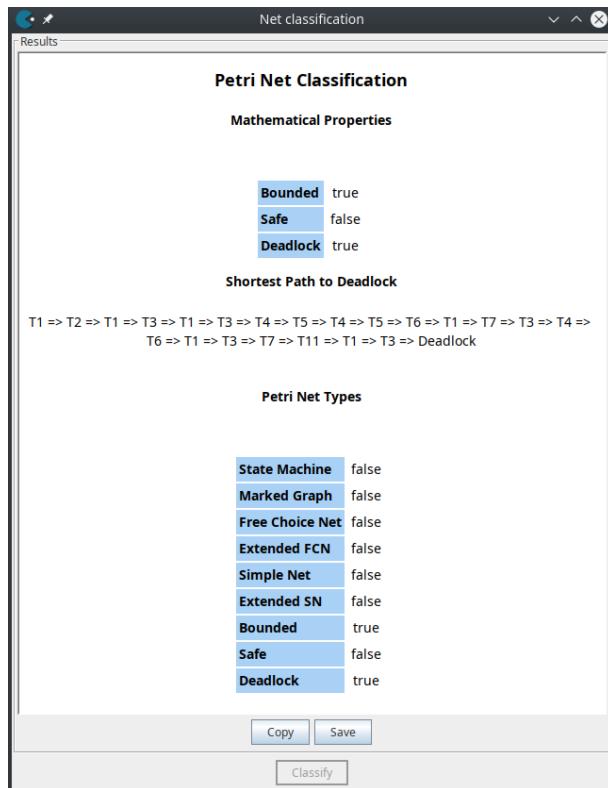


FIGURA A.6: Clasificación y propiedades de la red .

- **Análisis de Invariantes:** como se observa en la figura A.5 se encuentra en el submenú la opción *Invariant analysis* que nos permite realizar este análisis. Una vez seleccionada la opción, se abre una nueva ventana como muestra la figura A.7 en dónde se debe pulsar en *Analyse* para que el software haga dicho análisis y luego en *Save*.

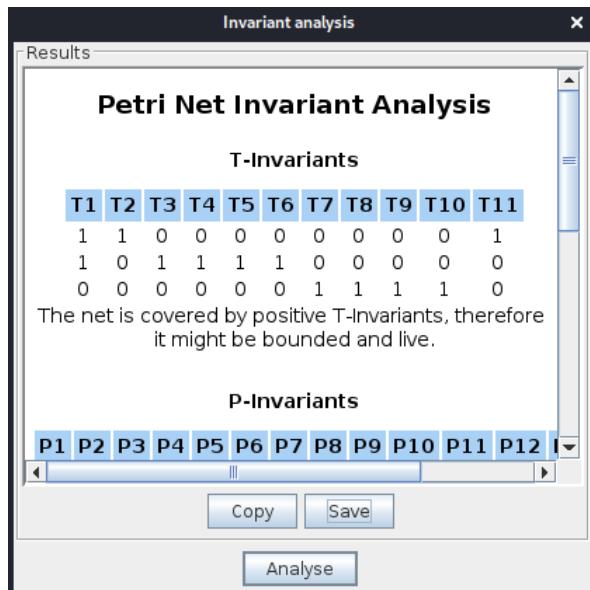
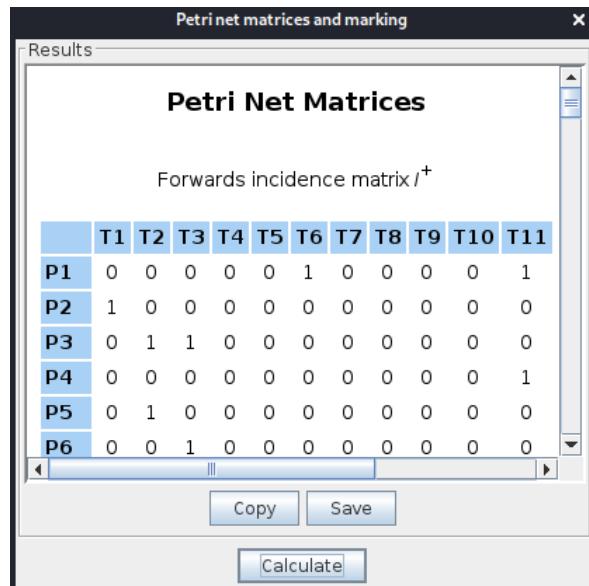


FIGURA A.7: Exportar invariantes.

- **Matrices:** como se observa en la figura A.5 se encuentra en el submenú la

opción *Matrices* que nos permite calcular las matrices asociadas a la red. Una vez seleccionada la opción, se abre una nueva ventana como muestra la figura A.8 en dónde se debe pulsar en *Calculate* para que el software haga los cálculos y luego en *Save*.



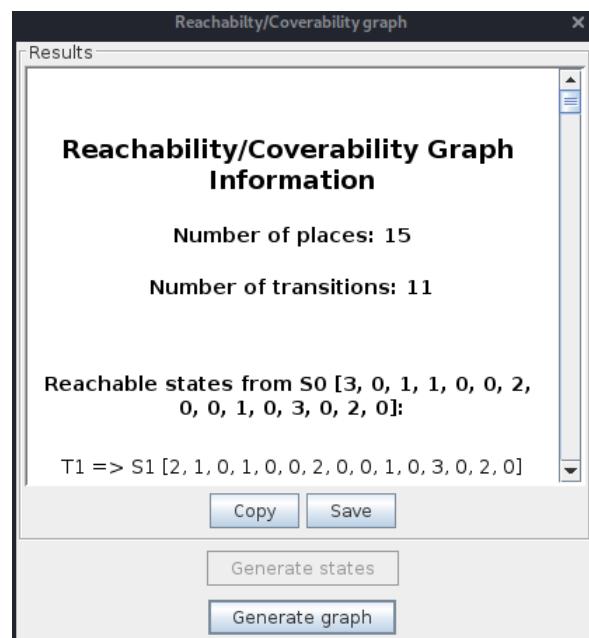
The screenshot shows a software window titled "Petri net matrices and marking". Inside, there's a section titled "Results" with a sub-section titled "Petri Net Matrices". Below this, it says "Forwards incidence matrix  $I^+$ ". A table is displayed with rows labeled P1 through P6 and columns labeled T1 through T11. The table values are as follows:

	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11
P1	0	0	0	0	0	1	0	0	0	0	1
P2	1	0	0	0	0	0	0	0	0	0	0
P3	0	1	1	0	0	0	0	0	0	0	0
P4	0	0	0	0	0	0	0	0	0	0	1
P5	0	1	0	0	0	0	0	0	0	0	0
P6	0	0	1	0	0	0	0	0	0	0	0

At the bottom of the window are buttons for "Copy", "Save", and "Calculate".

FIGURA A.8: Exportar matrices.

- **Grafo de Alcanzabilidad:** como se observa en la figura A.5 se encuentra en el submenú la opción *Reachability/Coverability graph* que nos permite generar el grafo con todos los estados alcanzables de la red. Una vez seleccionada la opción, se abre una nueva ventana como muestra la figura A.9 en dónde se debe pulsar en *Generate states* para que el software genere dicho análisis y luego en *Save*.



The screenshot shows a software window titled "Reachability/Coverability graph". Inside, there's a section titled "Results" with a sub-section titled "Reachability/Coverability Graph Information". It displays the following information:

- Number of places: 15
- Number of transitions: 11
- Reachable states from S0 [3, 0, 1, 1, 0, 0, 2, 0, 0, 1, 0, 3, 0, 2, 0]:

Below this, it shows a transition labeled "T1 => S1 [2, 1, 0, 1, 0, 0, 2, 0, 0, 1, 0, 3, 0, 2, 0]" with a dropdown arrow. At the bottom of the window are buttons for "Copy", "Save", "Generate states", and "Generate graph".

FIGURA A.9: Exportar grafo de alcanzabilidad.

- **Sifones y Trampas:** como se observa en la figura A.5 se encuentra en el submenú la opción *Siphons and traps* que nos permite calcular todos los sifones y trampas presentes en la red. Una vez seleccionada la opción, se abre una nueva ventana como muestra la figura A.10 en dónde se debe pulsar en *Analyse* para que el software realice el cálculo y luego en *Save*.

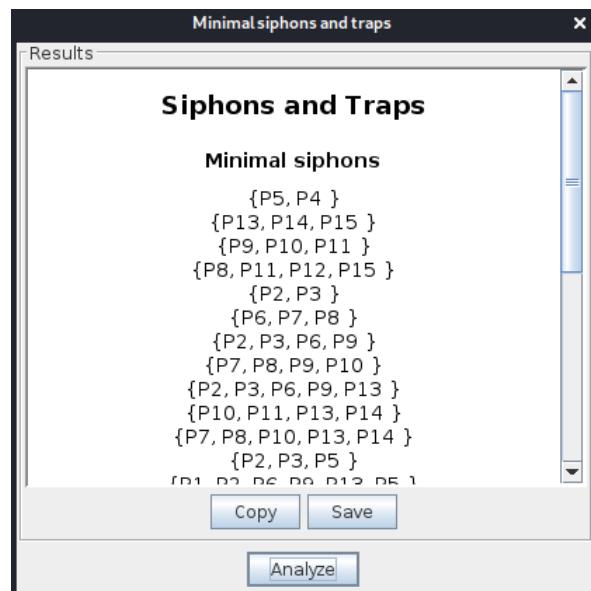


FIGURA A.10: Exportar sifones y trampas.

## Apéndice B

# Ejecución del algoritmo completa y detallada

En este apéndice, se detalla el procedimiento de ejecución de la última versión del algoritmo (Sección 3.5) para un caso específico.

### B.1. Ejecución del caso Panamá

Siguiendo los pasos mencionados en el apéndice A se carga la red Panamá previamente creada.

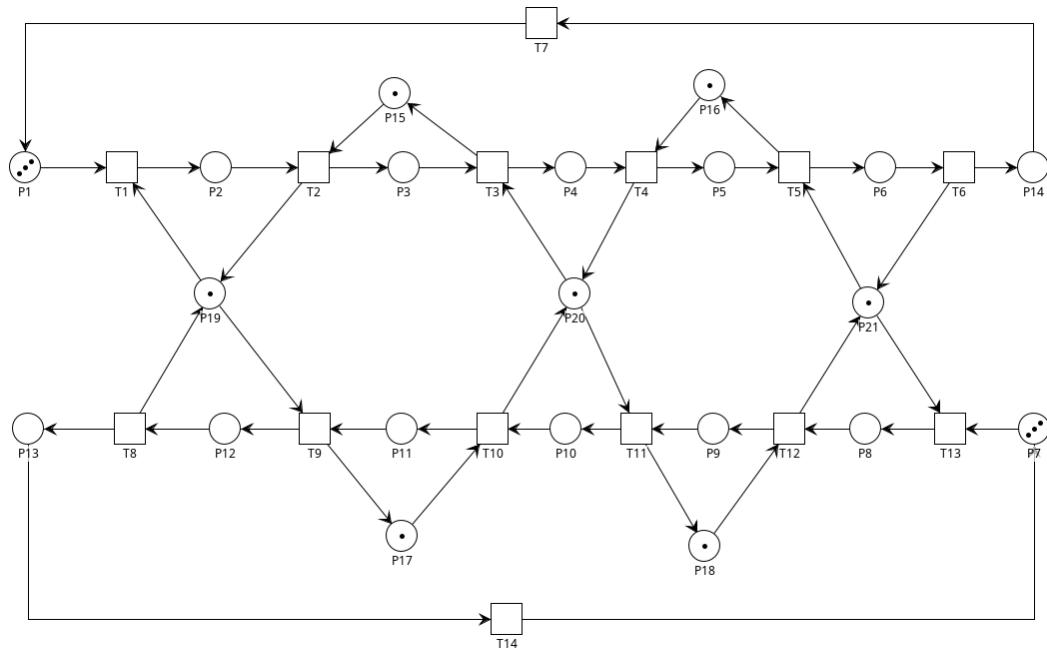


FIGURA B.1: RdP Panamá<sup>1</sup>

En primera instancia se verifica la presencia de deadlock como se muestra en la Figura A.6; al verificar que la red presenta deadlock **true** se prosigue con la extracción de los otros archivos necesarios:

- Análisis de invariantes.

<sup>1</sup>Figura adaptada del libro *An Algorithm for Deadlock Prevention Based on Iterative Siphon Control of Petri Net* [10].

- Matrices.
- Grafo de alcanzabilidad.
- Sifones y trampas.

Se realiza la ejecución del algoritmo con *Python v3*:

```
$ python3 algoritmo-v3.py
```

Se solicita el ingreso de los archivos exportados del Petrinator (.html). Siendo:

- g.html : Grafo de alcanzabilidad.
- m.html : Matrices
- s.html : Sifones y trampas.
- i.html : Análisis de invariantes.

El mismo arroja la cantidad de estados que presentan deadlock, el número de sifones vacíos asociados y los correspondientes supervisores para controlarlos cada uno con su respectivo id, marcado y transiciones input/output.

```
andres@salva:~/Escritorio/Bancodeprueba$ python3 algoritmo-v3.py
-----
Algoritmo para la solucion de deadlock para redes de Petri tipo S3PR
-----
Path del archivo de Estados (html): g.html
Path del archivo de Matrices I(html): m.html
Path del archivo de Sifones(html): s.html
Path del archivo de invariantes (html): i.html

Cantidad de estados con deadlock 2
Cantidad de sifones vacios: 3

Sifon a controlar: 12
Plazas del Sifon: ['P6', 'P10', 'P16', 'P18', 'P20', 'P21']
Marcado del supervisor 3
Transicion output: 1
Transicion output: 13
Transicion input: 5
Transicion input: 11

Sifon a controlar: 18
Plazas del Sifon: ['P6', 'P10', 'P11', 'P16', 'P18', 'P20', 'P21']
Marcado del supervisor 3
Transicion output: 1
Transicion output: 13
Transicion input: 5
Transicion input: 10
Transicion input: 11

Sifon a controlar: 19
Plazas del Sifon: ['P4', 'P12', 'P15', 'P17', 'P19', 'P20']
Marcado del supervisor 3
Transicion output: 1
Transicion output: 13
Transicion input: 3
Transicion input: 9
```

FIGURA B.2: Ejecución del primer análisis de la red.

Una vez seleccionado el supervisor a colocar, este se debe colocar manualmente. Como se muestra en la siguiente figura B.3, la plaza resaltada  $P_{22}$  es el supervisor agregado.

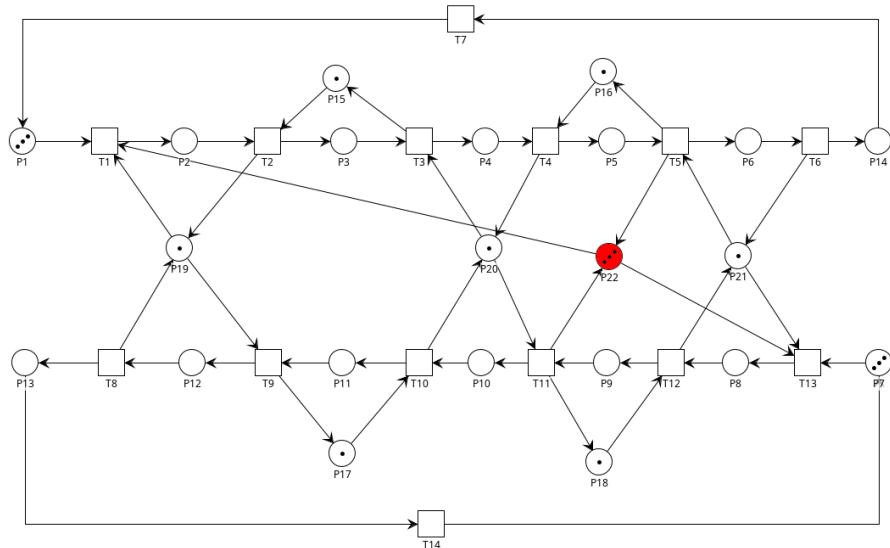


FIGURA B.3: Supervisor agregado.

Se debe verificar si el supervisor colocado soluciona el problema de deadlock realizando el análisis de la red. En caso de no ser así, sigue existiendo el deadlock, se deberán extraer nuevamente los archivos correspondientes a la nueva red.

Nuevamente se debe ejecutar el algoritmo, ingresando los archivos para realizar un nuevo análisis.

```
andres@salva:~/Escritorio/Bancodeprueba$ python3 algoritmo-v3.py
Algoritmo para la solucion de deadlock para redes de Petri tipo S3PR
-----
Path del archivo de Estados (html): g1.html
Path del archivo de Matrices I(html): m1.html
Path del archivo de Sifones(html): s1.html
Path del archivo de invariantes (html): i1.html

Cantidad de estados con deadlock 1
Cantidad de sifones vacios: 1

Sifon a controlar: 13
Plazas del Sifon: ['P4', 'P12', 'P15', 'P17', 'P19', 'P20']
Marcado del supervisor 3
Transicion output: 1
Transicion output: 13
Transicion input: 3
Transicion input: 9
```

FIGURA B.4: Segunda Iteración: análisis de la red con supervisores.

Como se puede observar en la Figura B.4 se presenta un único supervisor a colocar. Al colocarlo la red queda controlada, eliminando así todos los estados que presentaban deadlock.

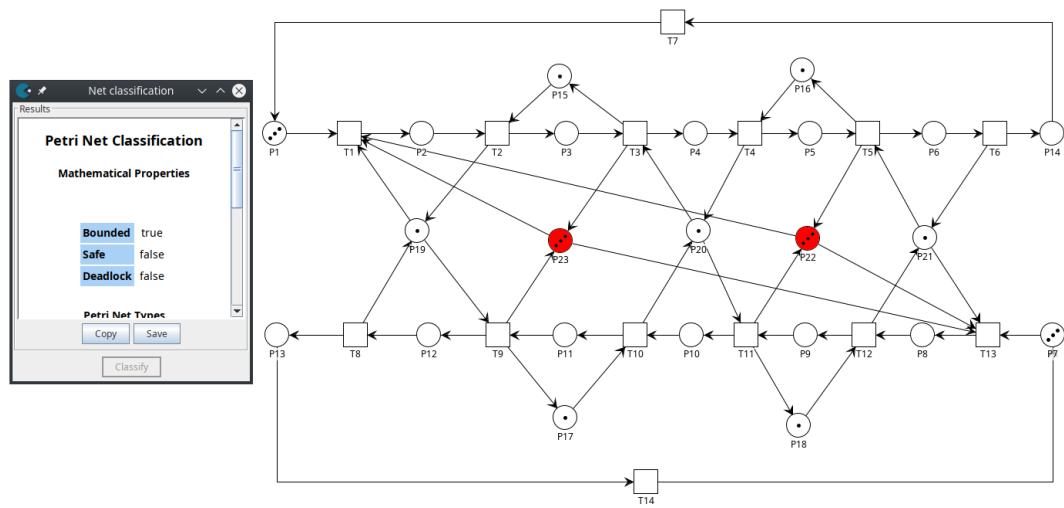


FIGURA B.5: Red resultante de agregar/eliminar arcos

# Bibliografía

- [1] Falko Bause **and** Pieter Kritzinger. *Stochastic Petri Nets - An Introduction to the Theory*. **november** 2013. ISBN: 3-528-15535-3.
- [2] G. W. Brams. *Las redes de Petri: teoría y práctica*. Tomo 1. Barcelona: Masson, 1986. ISBN: 8431103930 9788431103934.
- [3] Yufeng Chen **and others**. «Design of a Maximally Permissive Liveness- Enforcing Petri Net Supervisor for Flexible Manufacturing Systems». **in:** *Automation Science and Engineering, IEEE Transactions on* 8 (**may** 2011), **pages** 374–393. DOI: [10.1109/TASE.2010.2060332](https://doi.org/10.1109/TASE.2010.2060332).
- [4] Joaquín Rodríguez Felici **and** Leandro José Assón. «Editor, simulador y analizador de Redes de Petri.» Tesis de grado. UNC - Facultad de Ciencias Exactas Físicas y Naturales, 2017.
- [5] C. Jiang G. Liu **and** M. Zhou. «Two Simple Deadlock Prevention Policies for S<sup>3</sup>PR Based on Key-Resource/Operation-Place Pairs». **in:** *IEEE Transactions on Automation Science and Engineering* 7.4 (2010), **pages** 945–957. DOI: [10.1109/TASE.2010.2050059](https://doi.org/10.1109/TASE.2010.2050059).
- [6] Yi-Sheng Huang. «Design of deadlock prevention supervisors using Petri nets». **in:** *The International Journal of Advanced Manufacturing Technology* 35.3 (**december** 2007), **pages** 349–362. ISSN: 1433-3015. DOI: [10.1007/s00170-006-0708-y](https://doi.org/10.1007/s00170-006-0708-y). URL: <https://doi.org/10.1007/s00170-006-0708-y>.
- [7] Y. -. Huang **and others**. «Siphon-Based Deadlock Prevention Policy for Flexible Manufacturing Systems». **in:** *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans* 36.6 (2006), **pages** 1248–1256. DOI: [10.1109/TSMCA.2006.878953](https://doi.org/10.1109/TSMCA.2006.878953).
- [8] J. M. Colom J. Ezpeleta **and** J. Martínez. «A Petri net based deadlock prevention policy for flexible manufacturing systems». **in:** *IEEE Transactions on Robotics and Automation* 11.2 (1995), **pages** 173–184. DOI: [10.1109/70.370500](https://doi.org/10.1109/70.370500).
- [9] Danko Kezić, Nedjeljko Peric **and** Ivan Petrović. «An Algorithm for Deadlock Prevention Based on Iterative Siphon Control of Petri Net». **in:** *AUTOMATIKA: Journal for Control, Measurement, Electronics, Computing and Communications* (korema@fer.hr); Vol.47 No.1-2 (**january** 2006).
- [10] Danko Kezić, Nedjeljko Peric **and** Ivan Petrović. «An Algorithm for Deadlock Prevention Based on Iterative Siphon Control of Petri Net.» **in:** *AUTOMATIKA: Journal for Control, Measurement, Electronics, Computing and Communications*. 47.1-2 (2006).
- [11] Zhi Wu Li **and** Meng Chu Zhou. *Deadlock Resolution in Automated Manufacturing Systems: A Novel Petri Net Approach*. 1st. Springer Publishing Company, Incorporated, 2009. ISBN: 184882243X.

- [12] Zhiwu Li **and** Na Wei. «Deadlock control of flexible manufacturing systems via invariant-controlled elementary siphons of petri nets». in: *The International Journal of Advanced Manufacturing Technology* 33 (**may** 2007), **pages** 24–35. DOI: [10.1007/s00170-006-0452-3](https://doi.org/10.1007/s00170-006-0452-3).
- [13] GaiYun Liu **and** Kamel Barkaoui. «A survey of siphons in Petri nets». in: *Information Sciences* 363 (2016), **pages** 198–220. ISSN: 0020-0255. DOI: <https://doi.org/10.1016/j.ins.2015.08.037>. URL: <http://www.sciencedirect.com/science/article/pii/S0020025515006258>.
- [14] Gaiyun Liu **and others**. «Robustness of deadlock control for a class of Petri nets with unreliable resources». in: *Information Sciences* 235 (**june** 2013), **pages** 259–279. DOI: [10.1016/j.ins.2013.01.003](https://doi.org/10.1016/j.ins.2013.01.003).
- [15] H. Liu **and others**. «Transition Cover-Based Design of Petri Net Controllers for Automated Manufacturing Systems». in: 44.2 (2014), **pages** 196–208. DOI: [10.1109/TSMC.2013.2238923](https://doi.org/10.1109/TSMC.2013.2238923).
- [16] Jianchao Luo, ZhiQiang Liu **and** Mengchu Zhou. «A Petri Net Based Deadlock Avoidance Policy for Flexible Manufacturing Systems With Assembly Operations and Multiple Resource Acquisition». in: *IEEE Transactions on Industrial Informatics* PP (**october** 2018), **pages** 1–1. DOI: [10.1109/TII.2018.2876343](https://doi.org/10.1109/TII.2018.2876343).
- [17] Orlando Micolini **and others**. «Ecuación de estado generalizada para redes de Petri no autónomas y con distintos tipos de arcos». in: **october** 2016.
- [18] Julian Nonino, Carlos Pisetta **and** Orlando Micolini. «Desarrollo de IP cores con procesamiento de Redes de Petri Temporales para sistemas multicore en FPGA». **november** 2012. DOI: [10.13140/RG.2.2.10266.29127](https://doi.org/10.13140/RG.2.2.10266.29127).
- [19] Sanchez Figueroa Fernando Palma Méndez José Tomas Garrido Carrera Maria del Carmen **and** Quesada Arenciaba Alexis. *Programación Concurrente*. Editorial Paraninfo, 2003.
- [20] C.A. Petri. *Kommunikation mit Automaten*. Schriften des Rheinisch-Westfälischen Institutes für Instrumentelle Mathematik an der Universität Bonn. Technische Hochschule, Darmstadt., 1962. URL: <https://books.google.com.ar/books?id=NCZMvAEACAAJ>.
- [21] William Stallings. *Sistemas operativos: aspectos internos y principios de diseño*. 5ta. Madrid: Pearson Educación, S.A., 2005. ISBN: 978-84-205-5796-0.
- [22] Murat Uzam. «An Optimal Deadlock Prevention Policy for Flexible Manufacturing Systems Using Petri Net Models with Resources and the Theory of Regions». in: *Int. J. Adv. Manuf. Technol.* 19 (**january** 2002), **pages** 192–208. DOI: [10.1007/s001700200014](https://doi.org/10.1007/s001700200014).
- [23] Murat Uzam. «The use of Petri net reduction approach for an optimal deadlock prevention policy for flexible manufacturing systems». in: *The International Journal of Advanced Manufacturing Technology* 23 (**february** 2004), **pages** 204–219. DOI: [10.1007/s00170-002-1526-5](https://doi.org/10.1007/s00170-002-1526-5).
- [24] Murat Uzam, Zhiwu Li **and** Mengchu Zhou. «Identification and elimination of redundant control places in Petri net based liveness enforcing supervisors of FMS». in: *Int. J. Adv. Manuf. Tech.* 35 (**november** 2007), **pages** 150–168. DOI: [10.1007/s00170-006-0701-5](https://doi.org/10.1007/s00170-006-0701-5).

- [25] Murat Uzam **and** Mengchu Zhou. «Iterative synthesis of Petri net based deadlock prevention policy for flexible manufacturing systems». **in:** **volume 5. november** 2004, 4260–4265 vol.5. ISBN: 0-7803-8566-7. DOI: [10.1109/ICSMC.2004.1401200](https://doi.org/10.1109/ICSMC.2004.1401200).
- [26] N. Viswanadham, Y. Narahari **and** T. L. Johnson. «Deadlock prevention and deadlock avoidance in flexible manufacturing systems using Petri net models». **in:** *IEEE Transactions on Robotics and Automation* 6.6 (1990), **pages** 713–723. DOI: [10.1109/70.63257](https://doi.org/10.1109/70.63257).
- [27] YiSheng Huang **and others**. «A deadlock prevention policy for flexible manufacturing systems using siphons». **in:** *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164)*. **volume 1.** 2001, 541–546 vol.1. DOI: [10.1109/ROBOT.2001.932606](https://doi.org/10.1109/ROBOT.2001.932606).
- [28] Mi Zhao **and** Yifan. Hou. «An iterative method for synthesizing non-blocking supervisors for a class of generalized Petri nets using mathematical programming.» **in:** *Discrete Event Dynamic Systems* 23 (2013). DOI: [10.1007/s10626-011-0124-9](https://doi.org/10.1007/s10626-011-0124-9).
- [29] MengChu Zhou **and** Naiqi Wu. *System Modeling and Control with Resource-Oriented Petri Nets*. 1st. USA: CRC Press, Inc., 2009. ISBN: 1439808848.